クロック・データ・リカバリ回路の機能をディジタル・シミュレータで検証する

―Verilog HDLでアナログ要素をモデリング

デバイスの記事

在原栄一

ここでは，ミックスト・シグナル（アナログ・ディジタル混在）回路を，Verilog HDLシミュレータを使って検証する手法を紹介する．アナログHDLなどに対応する高価なシミュレータを用意しなくても，低コストの環境で回路のふるまいを確認できる．サンプルは，トランシーバLSIなどに使われているクロック・データ・リカバリ回路である．（編集部）

一般的なミックスト・シグナルLSIの機能検証手法を議論する場合，対象となる回路によって最適と思われる検証手段が異なります．例えば，A-D変換やD-A変換，PLL（phase-locked loop）回路，PLL回路を含んだ機能ブロック（今回，例として挙げたクロック・データ・リカバリ回路や周波数シンセサイザなど），電源回路，RFフロントエンドを含んだ機能ブロック（ミキサ回路を含んだ機能ブロックなど），LCDドライバ回路の検証において，一つの手法だけで対処するのは困難です．ただ，いずれの分野も検証時間のかかるアナログ部をどう取り扱うかが一つのポイントになります．

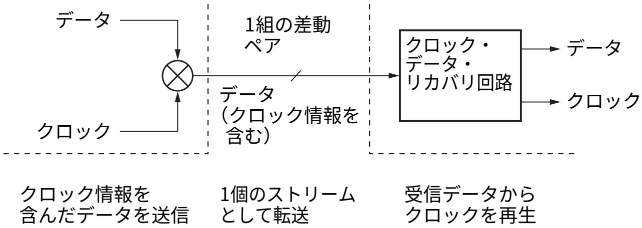


図1　CDRによるデータの送受信[1]
CDRによるデータの送受信では，受信データからクロックを再生する．IC外のクロック・スキューの問題を回避し，高速データ転送は実現できる．

今回はPLLをベースにしたクロック・データ・リカバリ回路を例として，Verilog HDLを用いた機能検証を行います．Verilog HDLをアナログ機能ブロックのビヘイビア・モデルとして採用する利点として，MATLABやC言語のモデルよりも最終的なレイアウト情報に近いことが挙げられます．A-DコンバータやD-Aコンバータのような離散時間的なアナログ機能ブロックのビヘイビア・モデルの記述にVerilog HDLを用いたケースは珍しくありませんでした．しかし，PLLのような連続時間的なアナログ機能ブロックのビヘイビア・モデルをVerilog HDLで記述する例は，それほど多くありませんでした．

以降では，対象となるクロック・データ・リカバリ回路を紹介し，検証方法やビヘイビア・モデルの内容，シミュレーション結果について解説します．

1　クロック・データ・リカバリ回路

現在の高性能通信システムでは，データ通信に1Gbps以上の送信速度が要求されるケースがしばしばあります．

● 高速データ伝送ではクロック・データ・リカバリが必須

一般的なシングルエンド標準I/O規格では，250MHz近辺の周波数でノイズ制約に直面すると言われています．これに対して差動標準I/O規格では，コモン・モードの除去によって，より少ないピン数で高速なデータ送信を可能にしました．しかし，差動標準I/O規格でも，1Gbps近くになると，クロック・スキューが問題となってきます．

KeyWord　ミックスト・シグナル，アナログ・ディジタル混在，Verilog HDL，アナログHDL，クロック・データ・リカバリ，CDR，VeritakWin，M系列，疑似乱数，ルンゲクッタ法

クロック・データ・リカバリ(CDR：clock data recovery)は，受信側においてデータ列からクロックを再生する方式であり，受信側IC外のクロック・スキューの問題を回避し，データとクロックの同期を容易にします(**図1**)．その反面，CDRではデータが基本的にNRZ(non return zero)形式であるため，データ列からVCO(voltage controlled oscillator；電圧制御発振器)の適正な発振中心周波数を求めるのにリファレンス周波数が必要となります．もしくは，DLL(delay locked loop)などを使用したより複雑な回路を付加しなければなりません．

なお，CDRでは，データおよび内部生成クロックのジッタ量(例えば，外来ノイズや，ループの持つ周期性やデバイス・ノイズなどに起因するノイズ)やロックするまでの時間などが，性能を決める指標になります．また，ユーザ側から見ると，リファレンス・クロックが不要になる，広い範囲のデータ・レート(クロック周波数)に対応できる，ローパス・フィルタ(LPF：low pass filter)用の外付けキャパシタが不要になる，VCOを内蔵する，といった特徴が望まれます(現在，半導体メーカから提供されている多くのカスタム品で，実現されつつある)．

● 10Gbpsに対応するサンプル回路

CDRは，現在，比較的多くの半導体メーカからカスタム品，もしくは機能ブロックとして提供されています．ここでは，CDRのアーキテクチャと回路について，数年前にIEEE Solid State Circuits誌に掲載された論文を例に説明します[(2)]．仕様は10Gbps(クロック周波数は5GHz)です．使用プロセスは0.18 μmプロセスのCMOS技術で，VCOを内蔵しています(実際の製品として考えた場合，レイアウト設計への負担も含めて，とても難しい仕様のように思われる)．このCDRの例を，今回の機能検証の対象とします．

図2がCDRのアーキテクチャです．VCOの周波数を高くしたくないので，10Gbpsのデータ・レートを実現するのに5GHzのクロックの半周期に対して1個のデータを割り当てています．また，ロック可能な周波数範囲を広げる目的，およびロックするまでの時間を短縮する目的で，位相検出・比較回路(ここではPDと呼ぶ)だけでなく，周波

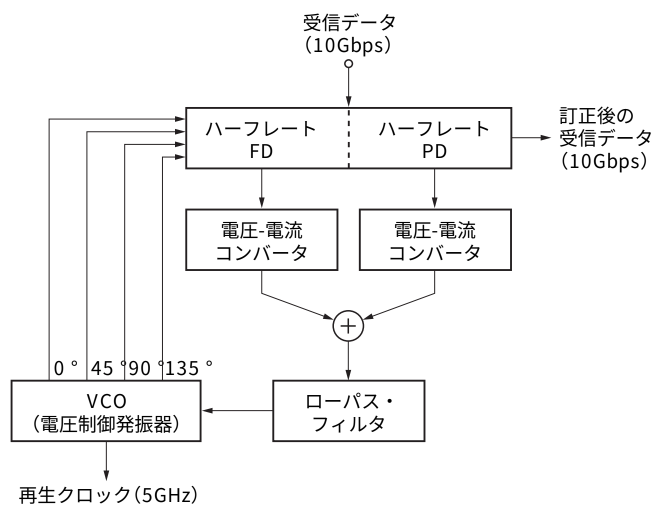


図2　CDRのアーキテクチャ例
5GHzクロックで10Gbpsのスループットを得られるアーキテクチャとなっている．

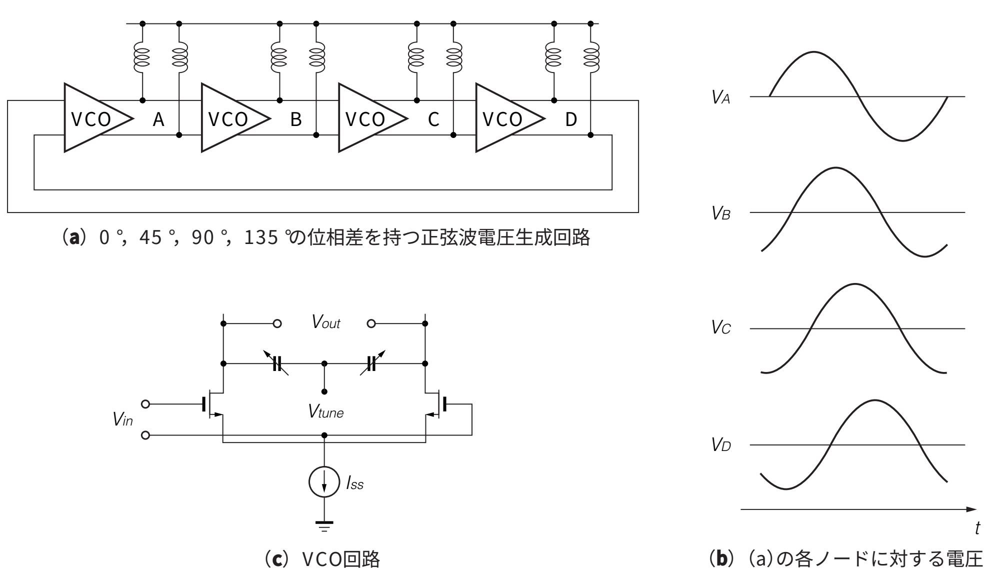


図3
VCOチェーン回路
4個のVCOを縦列接続することで，0°(deg)，45°，90°，135°の位相差を持つ正弦波電圧を生成している．

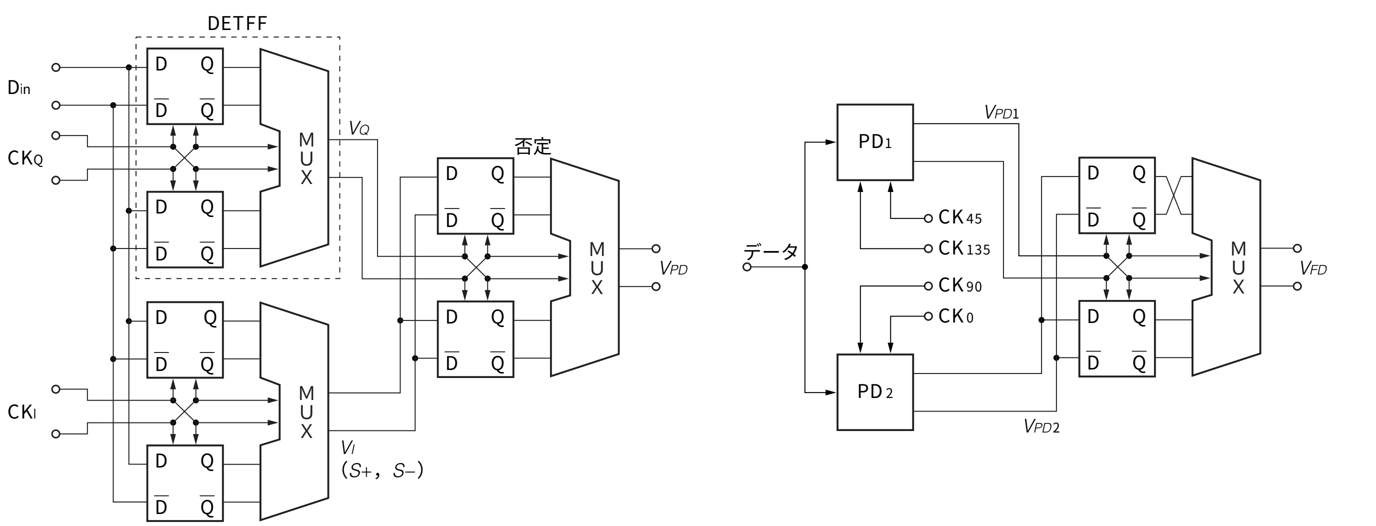
数検出・比較回路（FDと呼ぶ）を設けています．さらに，VCOで生成する4種類の位相差（0°，45°，90°，135°）を持つクロックを使用し，PDやFDの内部で，データの持つクロック情報とVCOで生成したクロックとの位相進みや位相遅れ，周波数の大小を判断します．

図3が，4種類の位相差を持つクロックを生成するVCOチェーン回路です．**図3(a)**は，**図3(c)**のように4個のLCタンク型VCOをチェーン接続した回路です．四つのノード（A，B，C，D）で45°ずつの位相差の正弦波電圧を生成します（**図3(b)**）．

同じように，**図4**はPD回路およびFD回路です．PDでは，受信データに対して内部生成クロックの位相が進んでいるか遅れているかの判定を行います．FDでは，2個のPDに45°ずつ位相差の異なる（VCOによる）内部生成クロックを入力し，周波数の大小の判定を行っています．これらの判定のようすについては，**図5**を参照してください．**図5(a)**で受信データD_{in}の論理状態に変化がない場合は，判定を保留しています．また，**図5(b)**では，周波数の大小判定をV_{PD1}とV_{PD2}の位相差で決めています（V_{FD}の符号）．

図6はチャージ・ポンプ回路（ここではCPと呼ぶ）で，**図4(b)**のV_{PD1}とV_{FD}を入力として$V_{PD1}-V_{FD}$を出力します．なお，出力$V_{PD1}-V_{FD}$については，**図5(b)**も参照ください．

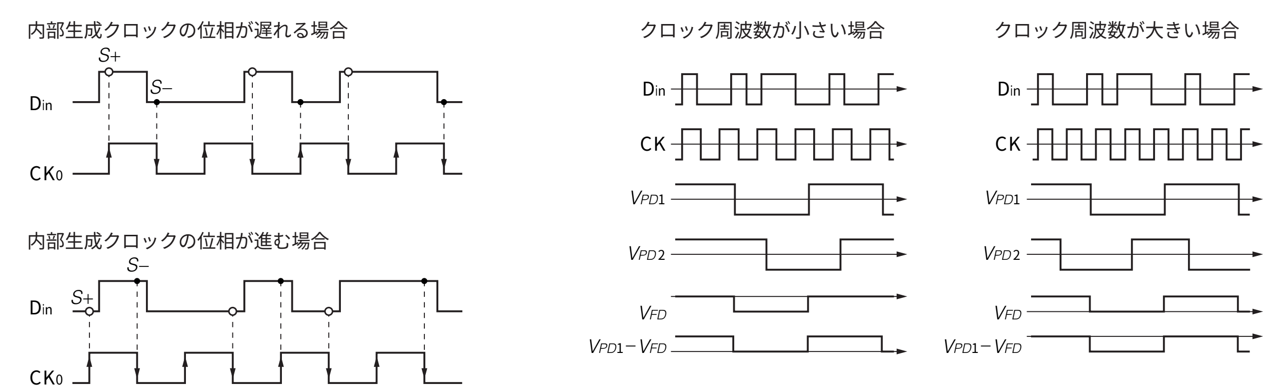
図7はローパス・フィルタです．**図6**のチャージ・ポン



(a) PD回路　(b) FD回路

図4　PD回路とFD回路

4種類の位相差のクロックを用いて，位相差や周波数差を検出する．



(a) 受信データと内部生成クロックとの位相関係の検出　(b) 受信データと内部生成クロックの周波数の関係の検出

図5　受信データと内部生成クロックの位相，および周波数の関係の検出

V_{PD1}は2値を，V_{FD}は3値を表現する．

プ回路出力である$V_{PD1}-V_{FD}$がこの回路の入力V_{in}に接続されます．さらに，このV_{in}はVCOの制御入力となり，**図3(c)**のV_{in}と接続されます．このフィルタによってPLL(二つの極と一つの零点を持つ)を構成します．なお，C_2はクロック・フィールド・スルーによる電圧リプルを抑制するための容量で，**図7**の構造はとても一般的です．

2 機能検証の方法

現在，比較的大規模なモデル，または機能検証時に多くのイベントを要するモデルのミックスト・シグナル・シミュレーションには，C言語やMATLAB，Verilog-A，Verilog-AMSなどが用いられています．システム検証からLSIの作り込みまでのどこに焦点を当てるかによって，これらの言語やツールが使い分けられています，

● ディジタル・シミュレータでアナログ要素の動作を模擬

今回は，Verilog HDL(Verilog-Aと対比されて，Veriog-Dとも呼ぶ)言語によって，上記で説明したCDR回路の機能検証(ミックスト・シグナル・シミュレーション)を行います．ミックスト・シグナル回路をディジタル回路用のVerilog HDLによって検証する意義は，以下の3点です(**図8**)．なお，Verilog HDLをコンパイル・実行するシミュレータとして，ここではWindows XP上で動作する，菅原システムズのVeritakWinを使用しました(下掲のコラム「VeritakWinとは？」を参照)．

- 合成対象となるディジタル回路はRTL(register transfer level)記述を使って機能検証を行いたい(ただし，今回の例では，合成対象となるディジタル回路は多くない)．
- アナログ回路においても，LSIの作り込みに近いモデルを使って機能検証を行いたい．つまり，レイアウト(でき上がったLSI)と同じ端子情報を備えたモデルで，機能検証を行いたい(Verilog HDLによるモデリングでは，アナログ端子を表現するために，要求される精度によってバス幅が決まる．例えばANALOG_INという端子は，Verilog HDLで`ANALOG_IN[15 : 0]`と表現する)．
- 用意しやすい(多くのユーザを有する)設計環境で，ミックスト・シグナルLSIの機能検証を短時間で行い，全体性能の向上を妨げる要因(アーキテクチャや回路)を特定したい(イベント数を最小限にとどめ，処理時間のかかるニュートン・ラプソン計算などの数値演算をできるだけ抑えることにより，SPICEやVerilog-Aと比べて検証時間を短縮する)．

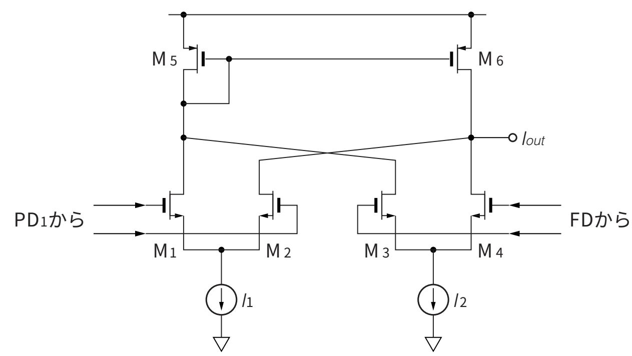


図6　チャージ・ポンプ回路
電圧-電流変換回路を兼ねている．

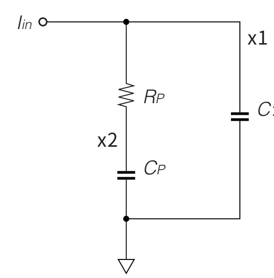


図7
ローパス・フィルタ
PLLを構成している．x1とx2は状態方程式で表現した場合の状態変数となるノードを示している．

● ジッタ量のピーク値とRMS値で評価

ここでは，もっとも代表的な特性値であるジッタ量のピーク値とRMS(root mean square)値を評価基準とします．ただし，ロック(収束)するまでの時間を1.5μs以内とします(つまり，ジッタ量の測定を開始後1.5μs後から始める)．

今，SONET(synchronous optical network；基幹伝送網の多重化通信体系)の規格に沿って，ピーク値を0.1UI

COLUMN《VeritakWinとは？》

VeritakWinは，菅原システムズ(http://japanese.sugawara-systems.com/)のVerilog HDLシミュレータで，編集，デバッグ，波形観測の機能を有しています．Windows XPとWindows 2000の上で動作します．なお，筆者が使用したバージョンは1.71Aです．

本シミュレータのユニークな特徴として，時間連続系で常微分方程式を解くためのライブラリを用意しています．SPICEなどのアナログ・シミュレータの過渡解析に相当するシミュレーションを行えます．

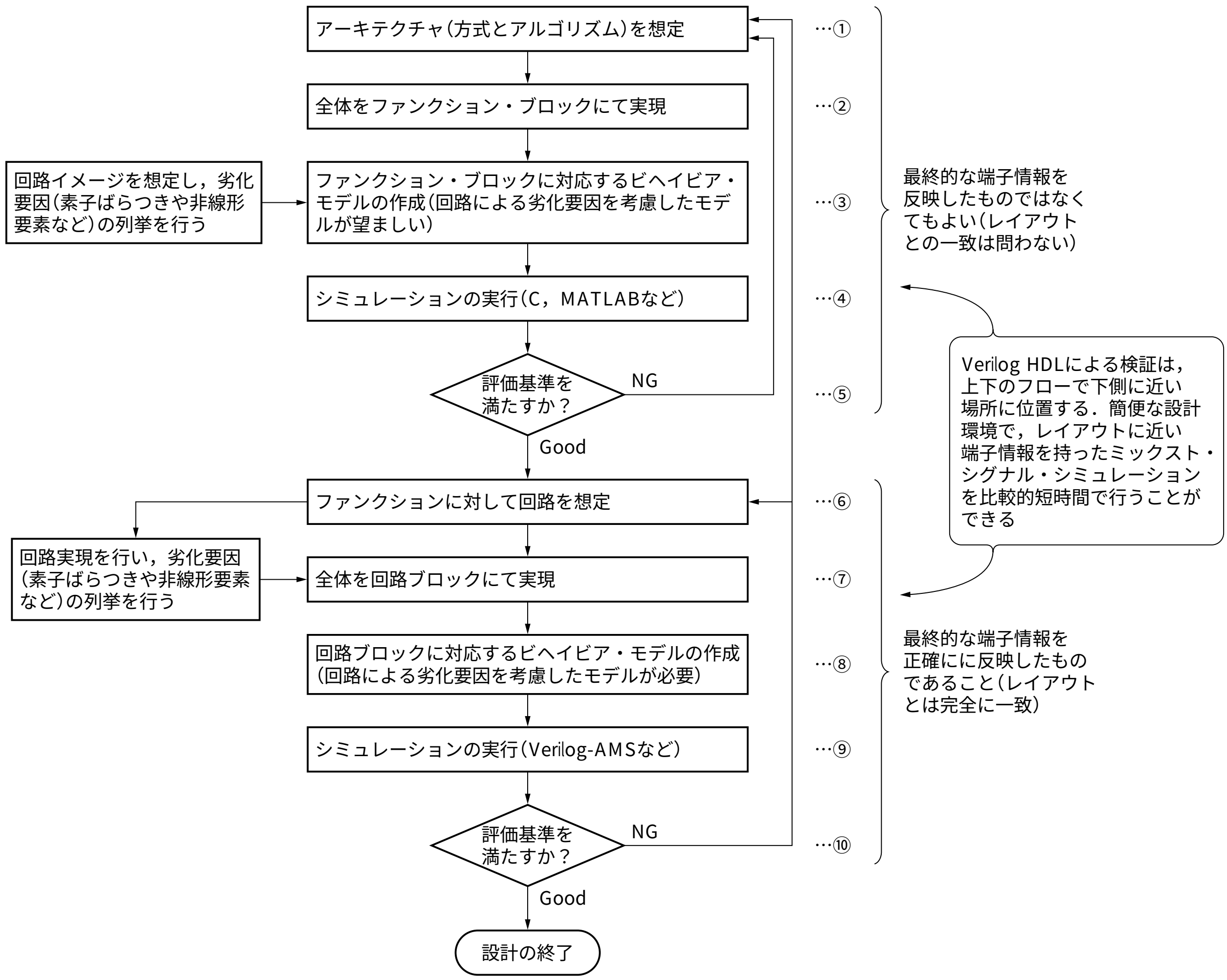


図8　ミックスト・シグナルLSIの設計・機能検証フロー例とVerilog HDLを使用する意義
アナログ・ビヘイビア・モデルを用いた設計フローの例を示す．SPICE系のアナログ・シミュレータでは多くの解析時間を要するため，ビヘイビア・モデルの導入を行い，段階を踏んだ階層設計・検証が現実的な解となる．

(unit intervals)以内，RMS値を0.01UI以内とします．今回，UI＝100psとするので，ピーク値は10psが，RMS値は1psが目標となります．なお，ピーク値とは偏差の最大値と最小値の差，RMS値とは偏差の2乗和の平方根を求めたものです．

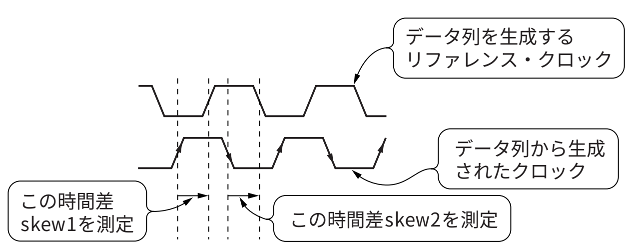


図9　ジッタ評価のためのスキュー(時間差)の測定
生成クロック・エッジからリファレンス・クロック・エッジまでの時間差の変動をジッタ量とする．

また，ジッタ量の規定ですが，入力データ列をもとに生成したクロック・エッジから，データ列を生成するリファレンス・クロックまでの時間差の変動を測定します(**図9**)．

3 モジュールの記述

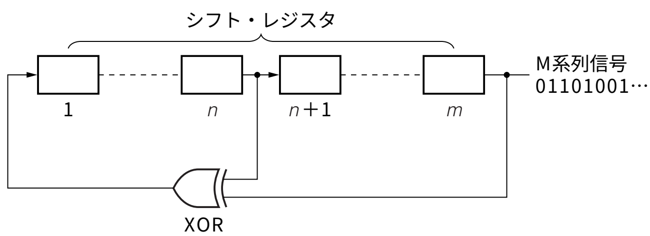
ここではシミュレーションを行うテストベンチ，CDR回路，入力データ生成回路，測定回路の順で，それぞれの記述を紹介します．

● テストベンチ：データ生成とジッタ測定の機能を用意

図10がテストベンチ(tb_top.v)のブロック図です．入力データ生成回路(dat_gen.v)で生成されたデータ列(Data)は，CDR回路(top.v)へと入力され，内部で生成されたク

COLUMN《M系列による疑似乱数の生成》

M系列信号(maximum length sequence)は2値("L"または"H")の周期信号です．具体的な回路は**図A-1**のとおりで，m個のDFF(Dフリップフロップ)と最低1個のXOR(排他的論理和)ゲートで最大2^m-1長の乱数を生成できます．また**表A-1**は，mの具体値と，最大2^m-1長の乱数を得るためのXOR入力のタップ位置を示しています．2^m-1長の乱数が繰り返されることから，疑似乱数列と呼ばれています[(3)]．



図A-1 M系列信号の生成回路
***m*個のDFFと最低1個のXORで最大2^m-1長のM系列信号(乱数)を生成できる**

表A-1 M系列信号における最大系列長とXOR入力タップ位置

シフト・レジスタ長(m)	タップ(n)	M系列長 2^m-1
15	14	32767
17	14	131071
18	11	262143
20	17	1048575
21	19	2097151
22	21	4194303
23	18	8388607
25	22	33554431
28	25	268435455
29	27	536870911
31	28	2147483647
33	20	8589934591

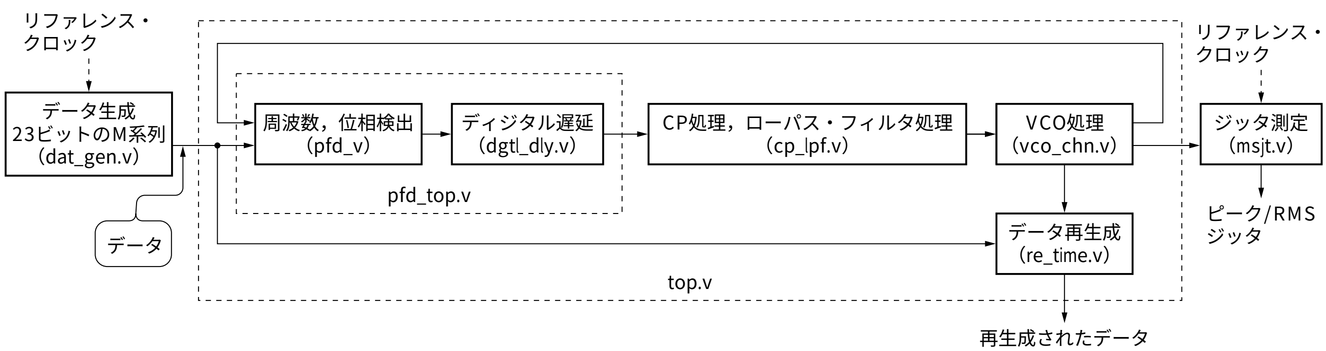


図10 CDR回路(top.v)のテストベンチ
テストベンチを示す．大きくトップ回路(CDR回路)，入力データ生成回路，ジッタ測定回路から構成される．

ロックで読み込まれ，出力(Re-timed Data)されます．また，測定回路(msjt.v)によって，VCO回路(vco_chn.v)で生成したクロックと，入力データ列を生成したリファレンス・クロックの時間差が測定され，ジッタ量が求められます．

リスト1がテストベンチ(tb_top.v)のソース・コードです．各回路で使用するパラメータは，`parameter`文によってここで設定し，`defparam`文で下の階層モジュールへと引き渡します．パラメータの意味については，該当するモジュールのところで説明します．また，時間単位と精度はともに1psとします．この1psは，ジッタ量(RMS値)の目標に対して余裕を持ってはいませんが，測定は可能です．なお，コードの簡単な説明を**リスト1**の中にコメント文として記してあります．

● CDR回路：階層化された五つのブロックからなる

再度，**図10**を参照してください．top.vがCDR回路のトップ階層(top)です．このtopモジュールと，topを構成するモジュール(module)について，以下に説明します．

1) topモジュール(top.v)

topモジュールはCDR回路のトップで，**図2**に対応しています．**リスト2**(p.121)がソース・コードです．テストベンチ上で指定されたパラメータ値を`parameter`文で受け取り，`defparam`文によって下の階層モジュールへと引き渡します．パラメータの意味については，該当するモジュールのところで説明します．

2) vco_chnモジュール(vco_chn.v)

vco_chnモジュールはVCO回路(VCOチェーン回路)で，**図3(a)**を実現したものです．**リスト3**(pp.122-123)が

リスト1　テストベンチ(tb_top.v)

テストベンチのパラメータ値のうち，ディジタル遅延値(TD_PD1とTD_FD)を2倍にしてシミュレーションした結果．図15に対して，ロック状態における大きなリプルを確認できる．

```
//CDR回路用テストベンチ
//Revised on 01:29 05/11/23

`timescale 1ps / 1ps

`include "top.v"
`include "dat_gen.v"
`include "msjt.v"

module tb_top;
  reg rst, ref_clk;
  reg strt_msjt, ckgen;
  reg msjt_clk;
  reg msjt_clk2;
  wire out_p, out_n, ckgen45_p, ckgen45_n, dat_p, dat_n;
  wire vdd, vss;
  wire [23:0] clf_out;

  integer ts_refclk;
  integer mcd;
  integer tmp_i1,tmp_i2;
  real tmp_r1,tmp_r2;

  //トップ回路(top)と測定回路(msjt)用パラメータ
  parameter TM_PRCS=0.001;//[nsec]，時間単位

  //トップ回路(top)用パラメータ
  parameter K_RTOI=524288.0;          //real変数からinteger変数へ変換係数
  parameter TD_PD1=0.3;               //[ns]， 遅延時間，PD用
  parameter TRF_PD1=0.1;              //[ns]， 立ち上がり・立ち下がり時間，PD用
  parameter K_DIV_PD1=20;   //(TD+TRF)/K_DIV値を処理サイクル
                                              (cycle)とする．PD用
  parameter TD_FD=0.5;                //[ns]， 遅延時間，FD用
  parameter TRF_FD=0.1;               //[ns]， 立ち上がり・立ち下がり時間，FD用
  parameter K_DIV_FD=20;              //(TD+TRF)/K_DIV値を処理サイクル
                                              (cycle)とする．FD用
  parameter CUR_CP_PD1=125.0;
                                      //[uA]，チャージ・ポンプの電流値，PD1用
  parameter RTO_B_A_PD1=1.0;
                                      //チャージ・ポンプの吸い込み電流と吐き出し
                                      電流の比率(Un-matchの影響度)，PD用
  parameter CUR_CP_FD=125.0;          //[uA]，チャージ・ポンプの電流値，FD用
  parameter RTO_B_A_FD=1.0;           //チャージ・ポンプの吸い込み電流と吐き出し
                                      電流の比率(Un-matchの影響度)，FD用
  parameter OFS_CUR_CP_FD=0.0;        //[uA]，中間レベル入力の
                                           チャージ・ポンプの電流値，FD用
  parameter CP=50.0;        //[pF]，LPFのキャパシタ値(フィルタ定数)
  parameter C2=2.0;         //[pF]，チャージ・ポンプ切り替り時の
                              フィードスルー緩和用　キャパシタ値．LPFで使用
  parameter RP=30.0;        //[Ohm]，ローパス・フィルタの抵抗値
                                                       (フィルタ定数)
  parameter TS_RK_LPF=0.001;          //[ns]，ルンゲクッタ処理の周期，LPF用
  parameter N_STP_INI_LPF=0.064;      //[ns]，ルンゲクッタ計算のステップ幅の
                                       初期値(LPF出力の更新周期の初期値)
  parameter RELERR_LPF=0.001;         //許容相対エラー項(ルンゲクッタ計算)
  parameter ABSERR_LPF=0.00001;       //許容絶対エラー項(ルンゲクッタ計算)
  parameter CYCLE_VCO=1;              //VCOの出力更新時間(=TM_PRCS*CYCLE_VCO)
  parameter THETA_INIT=0.0; //[rad]，VCOの初期位相

  //測定回路(msjt)用パラメータ
  parameter CYCLE_MJ=1;     //測定を行う際のスキップ数．1/CYCLE_MJが
                                         測定頻度を意味する
  parameter T_AVRG=300;               //[ns]，strt_msjtがL->H後のジッタ平均値
                                              を求めるための時間を指す
  parameter T_AVMJ=500;               //[ns]，ピーク・ジッタ値やRMSジッタ値を
                                                求めるための時間を指す
  parameter TS_CLK=0.2;               //[ns]，リファレンス・クロックの周期

  defparam I2.TM_PRCS=TM_PRCS;
  defparam I2.K_RTOI=K_RTOI;
  defparam I2.TD_PD1=TD_PD1;
  defparam I2.TRF_PD1=TRF_PD1;
  defparam I2.K_DIV_PD1=K_DIV_PD1;
  defparam I2.TD_FD=TD_FD;
  defparam I2.TRF_FD=TRF_FD;
  defparam I2.K_DIV_FD=K_DIV_FD;
  defparam I2.N_STP_INI_LPF=N_STP_INI_LPF;
  defparam I2.RELERR_LPF=RELERR_LPF;
  defparam I2.ABSERR_LPF=ABSERR_LPF;
  defparam I2.TS_RK_LPF=TS_RK_LPF;
  defparam I2.CUR_CP_PD1=CUR_CP_PD1;
  defparam I2.RTO_B_A_PD1=RTO_B_A_PD1;
  defparam I2.CUR_CP_FD=CUR_CP_FD;
  defparam I2.RTO_B_A_FD=RTO_B_A_FD;
  defparam I2.OFS_CUR_CP_FD=OFS_CUR_CP_FD;
  defparam I2.CP=CP;
  defparam I2.C2=C2;
  defparam I2.RP=RP;
  defparam I2.CYCLE_VCO=CYCLE_VCO;
  defparam I2.THETA_INIT=THETA_INIT;

  defparam I3.TM_PRCS=TM_PRCS;
  defparam I3.CYCLE_MJ=CYCLE_MJ;
  defparam I3.T_AVRG=T_AVRG;
  defparam I3.T_AVMJ=T_AVMJ;
  defparam I3.TS_CLK=TS_CLK;

  //---ネットリスト
  assign vdd=1'b1;
  assign vss=1'b0;
  assign ckgen45=(ckgen45_p==ckgen45_n)? 1'b0 :
                                     ckgen45_p;
  dat_gen I1 (rst,ref_clk,dat_p,dat_n);
  top I2 (rst, dat_p, dat_n, out_p, out_n,
         ckgen45_p,ckgen45_n, clf_out, vdd, vss);
  msjt I3 (strt_msjt, ref_clk, ckgen45);

  initial begin
    tmp_r1=TS_CLK/TM_PRCS;
    ts_refclk=tmp_r1;
                    //入力データ生成用リファレンス
                                        ・クロック周期
    mcd=$fopen("rslt.dat");
                    //結果書き込み用ファイルの指定
    rst=1'b1;       //リセット有効
    strt_msjt=1'b0;
                    //測定無効

    #100 rst=1'b0;   //リセット解除
    #1200000 strt_msjt=1'b1;
                    //測定有効
    #810000 begin
      $fdisplay(mcd, "ジッタ量---平値：%d",I3.rslt_av);
      $fdisplay(mcd, "ジッタ量---正側ピーク値：%d、
                                負側ピーク値：%d"
                 ,I3.mj_pk_max, I3.mj_pk_min);
      tmp_r1=I3.mj_rms2_i;
      tmp_r2=I3.hit_jtrms;
      $fdisplay(mcd, "ジッタ量---RMSの2乗値：%g",
                                  (tmp_r1/tmp_r2));
      $fdisplay(mcd, "VCO生成クロック半周期
                                    ---最小値：%d,
               最大値：%d", I3.tsh_min_gen_clk,
                                .tsh_max_gen_clk);
      $fdisplay(mcd, "ルンゲクッタ計算で許容誤差を
                              満足しなかった回数：
                          %d,I2.I2.hit_min_h);
      $fclose(mcd);
      $finish;
    end

  end

  always begin
          //入力データ生成用リファレンス・クロックの生成
    ref_clk=1'b0; #(ts_refclk/2);
    ref_clk=1'b1; #(ts_refclk/2);
  end

  always begin
          //VCOチェーン回路制御電圧値を書き込むための
                                       クロック生成
    msjt_clk=1'b0; #500;
    msjt_clk=1'b1; #500;
  end

  always @ (msjt_clk) begin
          //VCOチェーン回路入力(制御)電圧値の書き込み
    $fdisplay(mcd, "%d     %g",$time, I2.I3.vin_rl);
                    //1行目は時間，2行目はVCO入力
  end

endmodule
```

ソース・コードです．機能は**リスト3**の中にも簡単な説明を記していますが，入力電圧を**図11**(p.123)の関係に基づいて(4次多項式近似により)45 °ずつ位相の異なる四つのクロックへと変換します．また，パラメータのTM_PRCSはシミュレーション時間単位(`timescaleで与えたのと同じ値)，CYCLE_VCOはスキップ数，CYCLE_VCO*TM_PRCS値は出力更新期間となります．それから，THETA_INITは発振出力の初期位相，K_RTOIは内部変数(実数)と信号(整数表現)の比(係数)です．なお，本来，本モジュールはデバイス・ノイズを考慮すべきブロックですが，今回は煩雑さを避ける目的からモデリングを省略しています．また，**図3(a)**以外の端子として，**リスト3**ではリセット信号(rst)と電源(vdd, vss)端子を追加しています(最終的なレイアウト情報に配慮した)．

3) pfd_top モジュール(pfd_top.v)

図10に示すように，pfd_topモジュールはPD回路とFD回路を記述したpfdモジュールと，ディジタル遅延回路を記述したdgtl_dlyモジュールから構成されます．ただし，

リスト2 topモジュール(top.v)

```
//CDR回路(トップ回路)
//Revised on 00:24 05/10/08

`include "pfd_top.v"
`include "cp_lpf.v"
`include "vco_chn.v"
`include "re_time.v"

module top (rst, datp, datn, outp, outn, ckgenp, ckgenn,
                                       clf_out, vdd, vss);
  input rst, datp, datn;
  input vdd, vss;
  output outp, outn;
  output ckgenp, ckgenn;//生成クロックで，データを生成する
      リファレンス・クロックとは90 °位相が異なるもの(clk_gen45)
  output [23:0] clf_out;
  wire pd1p, pd1n, fdp, fdn;
  wire ck0p, ck0n, ck45p, ck45n, ck90p, ck90n, ck135p,
                                                   ck135n;
  wire [23:0] clf_out;

  //I1～I3(pfd_top, cp_lpf, vco_chn)用パラメータ
  parameter TM_PRCS=0.001;
        //[ns], 時間精度

  //I2とI3(cp_lpf, vco_chn)用パラメータ
  parameter K_RTOI=524288.0;
        //real変数からinteger変数へ変換係数

  //I1(pfd_top)用パラメータ
  parameter TD_PD1=1.0;
        //[ns], 遅延時間，PD用
  parameter TRF_PD1=0.2;
        //[ns], 立ち上がり・立ち下がり時間，PD用
  parameter K_DIV_PD1=50;
        //(TD+TRF)/K_DIV値を処理サイクル(cycle)とする．PD用
  parameter TD_FD=1.0;   //[ns], 遅延時間，FD用
  parameter TRF_FD=0.2;
        //[ns], 立ち上がり・立ち下がり時間，FD用
  parameter K_DIV_FD=50;
        //(TD+TRF)/K_DIV値を処理サイクル(cycle)とする．FD用

  //I2(cp_lpf)用パラメータ
  parameter CUR_CP_PD1=5.0;
        //[uA], チャージ・ポンプの電流値，PD用
  parameter RTO_B_A_PD1=1.0;
        //チャージ・ポンプの吸い込み電流と吐き出し電流の比率
                                 (Un-matchの影響度), PD用
  parameter CUR_CP_FD=5.0;
        //[uA], チャージ・ポンプの電流値，FD用
  parameter RTO_B_A_FD=1.0;
        //チャージ・ポンプの吸い込み電流と吐き出し電流の比率
                                 (Un-matchの影響度), FD用
  parameter OFS_CUR_CP_FD=0.0;
        //[uA], 中間レベル入力でのチャージ・ポンプの電流値，FD用
  parameter CP=8.0;
        //[pF], LPFのキャパシタ値(フィルタ定数)
  parameter C2=0.2;
        //[pF], チャージ・ポンプ切り替え時のフィードスルー
                              緩和用キャパシタ値，LPFで使用
  parameter RP=1000.0;
        //[Ohm], LPFの抵抗値(フィルタ定数)
  parameter TS_RK_LPF=0.001;
        //[ns], ルンゲクッタ処理の周期，LPF用
  parameter N_STP_INI_LPF=0.1;
        //[ns], ルンゲクッタ計算のステップ幅の初期値
                                   (CP出力の更新周期の初期値)
  parameter RELERR_LPF=0.001;
        //許容相対エラー項(ルンゲクッタ計算)
  parameter ABSERR_LPF=0.00001;
        //許容絶対エラー項(ルンゲクッタ計算)

  //I3(vco_chn)用パラメータ
  parameter CYCLE_VCO=1;
        //VCOでの出力更新時間(=TM_PRCS*CYCLE_VCO)
  parameter THETA_INIT=0.0;
        //[rad], VCOでの初期位相

  defparam I1.TM_PRCS=TM_PRCS;
  defparam I1.TD_PD1=TD_PD1;
  defparam I1.TRF_PD1=TRF_PD1;
  defparam I1.K_DIV_PD1=K_DIV_PD1;
  defparam I1.TD_FD=TD_FD;
  defparam I1.TRF_FD=TRF_FD;
  defparam I1.K_DIV_FD=K_DIV_FD;

  defparam I2.TM_PRCS=TM_PRCS;
  defparam I2.K_RTOI=K_RTOI;
  defparam I2.CUR_CP_PD1=CUR_CP_PD1;
  defparam I2.RTO_B_A_PD1=RTO_B_A_PD1;
  defparam I2.CUR_CP_FD=CUR_CP_FD;
  defparam I2.RTO_B_A_FD=RTO_B_A_FD;
  defparam I2.OFS_CUR_CP_FD=OFS_CUR_CP_FD;
  defparam I2.CP=CP;
  defparam I2.C2=C2;
  defparam I2.RP=RP;
  defparam I2.TS_RK=TS_RK_LPF;
  defparam I2.N_STP_INI=N_STP_INI_LPF;
  defparam I2.RELERR=RELERR_LPF;
  defparam I2.ABSERR=ABSERR_LPF;

  defparam I3.TM_PRCS=TM_PRCS;
  defparam I3.CYCLE_VCO=CYCLE_VCO;
  defparam I3.THETA_INIT=THETA_INIT;
  defparam I3.K_RTOI=K_RTOI;

  //---ネットリスト
  assign ckgenp=ck45p;
  assign ckgenn=ck45n;
  pfd_top I1 (rst, datp, datn, ck0p, ck0n, ck45p, ck45n,
                          ck90p, ck90n, ck135p, ck135n,
                        pd1p, pd1n, fdp, fdn, vdd, vss);
  cp_lpf I2 (rst, pd1p, pd1n, fdp, fdn, clf_out, vdd,
                                                   vss);
  vco_chn I3 (rst, clf_out, ck0p, ck0n, ck45p, ck45n,
                 ck90p, ck90n, ck135p, ck135n, vdd, vss);
  re_time I4 (rst, ck45p, ck45n, datp, datn, outp, outn,
                                             vdd, vss);

endmodule
```

リスト3　vco_chn モジュール(vco_chn.v)

```
//VCOチェーン回路
//Revised on 01:23 05/09/14
//vin, vout0, vout45, vout90, vout135 :: signed-integer,
4bit-over + 20bit-core

module
vco_chn(rst,vin,out0p,out0n,out45p,out45n,out90p,out90n,
                              out135p,out135n,vdd,vss);
  input rst;
  input [23:0] vin;
  input vdd,vss;
  output  out0p,out0n,out45p,out45n,out90p,out90n,
                                      out135p,out135n;
  reg  out0p,out0n,out45p,out45n,out90p,out90n,
                                      out135p,out135n;
  reg [31:0] tmp32b_1,tmp32b_2;
  reg [23:0] vin2, vin24b_lmtH, vin24b_lmtL;

  integer cycle;
  real tmp_r1, tmp_r2, PI2, theta, dlt_t;
  real tmp_r3a, tmp_r3b, tmp_r3c, tmp_r3d, tmp_r4a,
                             tmp_r4b, tmp_r4c, tmp_r4d;
  real a0, a1, a2, a3, a4, frq, vin_rl;

  parameter TM_PRCS=0.001;
                      //[ns], 時間単位
  parameter CYCLE_VCO=1;
                      //VCOの出力更新期間をTM_PRCS*CYCLE_
                                        VCOで与える
  parameter THETA_INIT=0.0;
                      //[rad], 初期位相
  parameter K_RTOI=524288.0;
                      //整数値と内部計算用実数値の変換比率
                      //整数2^20に対して, 実数2.0が対応する
  initial begin
    PI2 = 6.283185307179586476925286766559O1;
    theta=THETA_INIT;
    dlt_t=TM_PRCS*CYCLE_VCO;
    cycle=CYCLE_VCO;

    vin24b_lmtH=24'h0E_6666;
    vin24b_lmtL=24'h00_0000;
    a0=4.6984;//a0～a4 : 4次多項式近似の係数。
    a1=0.2565;
    a2=-0.9054;
    a3=1.6146;
    a4=-0.548;
  end

  always #(cycle) begin
    if (rst) begin
      theta=THETA_INIT;
      out0p = 1'b0;
      out0n = ~out0p;
      out45p = 1'b0;
      out45n = ~out45p;
      out90p = 1'b0;
      out90n = ~out90p;
      out135p = 1'b0;
      out135n = ~out135p;
    end else begin
      //シングルエンド入力vinに対するリミッタ
      if(vin[23]==1'b0)
        vin2=(vin > vin24b_lmtH)? vin24b_lmtH : vin;
      else
        vin2=vin24b_lmtL;

      //整数から実数へ変換。
      tmp32b_1={ {8{vin2[23]} }, vin2};
      tmp_r1=$itor(tmp32b_1);
      vin_rl=tmp_r1/K_RTOI;

      //入力電圧を, 周波数, 位相へ変換
      frq = a0
        +a1*vin_rl
        +a2*(vin_rl*vin_rl)
        +a3*(vin_rl*vin_rl*vin_rl)
        +a4*(vin_rl*vin_rl*vin_rl*vin_rl);
      tmp_r2=theta +dlt_t*PI2*frq;
      theta=(tmp_r2 > PI2)? (tmp_r2-PI2) : tmp_r2;

      //位相(四つの位相)を電圧に変換し, ディジタル化する
      tmp_r3a=$sin(theta);
      tmp_r3b=$sin(theta-(PI2/8.0));
      tmp_r3c=$sin(theta-(PI2/4.0));
```

リスト4　pfd_top モジュール(pfd_top.v)

```
//位相・周波数検出回路 + ディジタル遅延回路
//Revised on 22:26 05/09/25
//pd1_outp, pd1_outn, fd_outp, fd_outn :: 1bit-data

`include "pfd.v"
`include "dgtl_dly.v"

module pfd_top (rst, datp, datn, ck0p, ck0n, ck45p,
        ck45n, ck90p, ck90n, ck135p, ck135n,  pd1_outp,
                 pd1_outn, fd_outp, fd_outn, vdd, vss);
  input rst, datp, datn, ck0p, ck0n, ck45p, ck45n,
                         ck90p, ck90n, ck135p, ck135n;
  input vdd, vss;
  output pd1_outp, pd1_outn, fd_outp, fd_outn;
  wire pd1_outp, pd1_outn, pd1_outp1, pd1_outn1;
  wire fd_outp, fd_outn, fd_outp1, fd_outn1;

  //I2～I5(dgtl_dly)用パラメータ
  parameter TM_PRCS=0.001; //[ns], 時間単位

  //I2とI3(two dgtl_dly)用パラメータ
  parameter TD_PD1=0.2;    //[ns], 遅延時間
  parameter TRF_PD1=0.2;   //[ns], 立ち上がり・立ち下がり時間
  parameter K_DIV_PD1=50;
       //(TD+TRF)/K_DIV値を処理サイクル(cycle)とする, PD用

  //I4とI5(two dgtl_dly)用パラメータ
  parameter TD_FD=0.2;     //[ns], 遅延時間
  parameter TRF_FD=0.2;    //[ns], 立ち上がり・立ち下がり時間
  parameter K_DIV_FD=50;
       //(TD+TRF)/K_DIV値を処理サイクル(cycle)とする, FD用

  defparam I2.TM_PRCS=TM_PRCS;
  defparam I2.TD=TD_PD1;
  defparam I2.TRF=TRF_PD1;
  defparam I2.K_DIV=K_DIV_PD1;

  defparam I3.TM_PRCS=TM_PRCS;
  defparam I3.TD=TD_PD1;
  defparam I3.TRF=TRF_PD1;
  defparam I3.K_DIV=K_DIV_PD1;

  defparam I4.TM_PRCS=TM_PRCS;
  defparam I4.TD=TD_FD;
  defparam I4.TRF=TRF_FD;
  defparam I4.K_DIV=K_DIV_FD;

  defparam I5.TM_PRCS=TM_PRCS;
  defparam I5.TD=TD_FD;
  defparam I5.TRF=TRF_FD;
  defparam I5.K_DIV=K_DIV_FD;

  //ネットリスト
  pfd I1 (rst, datp, datn, ck0p, ck0n, ck45p, ck45n,
                        ck90p, ck90n, ck135p, ck135n,
    pd1_outp1, pd1_outn1, fd_outp1, fd_outn1, vdd, vss);
  dgtl_dly I2 (pd1_outp1,pd1_outp,vdd,vss);
  dgtl_dly I3 (pd1_outn1,pd1_outn,vdd,vss);
  dgtl_dly I4 (fd_outp1,fd_outp,vdd,vss);
  dgtl_dly I5 (fd_outn1,fd_outn,vdd,vss);

endmodule
```

```
        tmp_r3d=$sin(theta-(PI2*3/8.0));
        tmp_r4a=tmp_r3a*K_RTOI;
        tmp_r4b=tmp_r3b*K_RTOI;
        tmp_r4c=tmp_r3c*K_RTOI;
        tmp_r4d=tmp_r3d*K_RTOI;
        tmp32b_2=$rtoi(tmp_r4a);
        out0p = (tmp32b_2[31]==1'b0)? 1'b1: 1'b0;
        out0n = ~out0p;
        tmp32b_2=$rtoi(tmp_r4b);
        out45p = (tmp32b_2[31]==1'b0)? 1'b1: 1'b0;
        out45n = ~out45p;
        tmp32b_2=$rtoi(tmp_r4c);
        out90p = (tmp32b_2[31]==1'b0)? 1'b1: 1'b0;
        out90n = ~out90p;
        tmp32b_2=$rtoi(tmp_r4d);
        out135p = (tmp32b_2[31]==1'b0)? 1'b1: 1'b0;
        out135n = ~out135p;
      end
    end

  endmodule
```

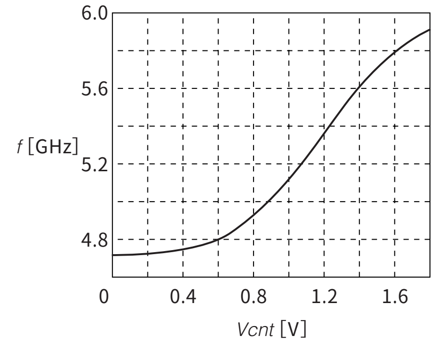


図11　vco_chn.v モジュールにおける入力電圧と出力周波数の関係

Vco_chn.v モジュールでは，4次多項式によって近似し，本図の関係を実現している．

リスト5　pfd モジュール(pfd.v)

```
//位相・周波数検出回路
//Revised on 03:11 05/09/25
//pd1_outp,pd1_outn,fd_outp,fd_outn :: 1bit-data

module
pfd(rst,datp,datn,ck0p,ck0n,ck45p,ck45n,ck90p,ck90n,
       ck135p,ck135n,pd1_outp,pd1_outn,fd_outp,fd_outn,
                                              vdd,vss);
  input rst,datp,datn,ck0p,ck0n,ck45p,ck45n,ck90p,ck90n,
       ck135p,ck135n;
  input vdd,vss;
  output pd1_outp, pd1_outn, fd_outp, fd_outn;
  wire pd1_outp, pd1_outn, fd_outp, fd_outn;
  wire dat,ck0,ck45,ck90,ck135;
  wire pd1_out;
  wire [1:0] fd_out1;

  assign dat=(datp==~datn)? datp : 1'b0;
  assign ck0=(ck0p==~ck0n)? ck0p : 1'b0;
  assign ck45=(ck45p==~ck45n)? ck45p : 1'b0;
  assign ck90=(ck90p==~ck90n)? ck90p : 1'b0;
  assign ck135=(ck135p==~ck135n)? ck135p : 1'b0;
  assign pd1_outp=pd1_out;        //差動正側
  assign pd1_outn=~pd1_outp;      //差動負側
  assign fd_outp=fd_out1[1];      //差動正側
  assign fd_outn=fd_out1[0];      //差動負側

  fd I1 (rst,dat,ck0,ck45,ck90,ck135,pd1_out,fd_out1);
endmodule

//周波数検出
module fd(rst,dat,ck0,ck45,ck90,ck135,pd1_out,fd_out);
  input rst,dat,ck0,ck45,ck90,ck135;
  output [1:0] fd_out;
  output pd1_out;
  reg [1:0] fd_out,tmp_2b;
  wire pd1_out,pd2_out;

  pd I1 (rst,dat,ck45,ck135,pd1_out);
  pd I2 (rst,dat,ck0,ck90,pd2_out);

  always @ (pd1_out or posedge rst) begin
    if(rst==1'b1)
      fd_out <= 2'b00;
    else begin
      if(pd1_out==1'b1)
        tmp_2b=(pd2_out==1'b1)? 2'b10 : 2'b11;
      else
        tmp_2b=(pd2_out==1'b1)? 2'b01 : 2'b11;
      fd_out <= tmp_2b;
    end
  end
endmodule

//位相検出
module pd(rst,dat,ck0,ck1,pd_out);
  input rst,dat,ck0,ck1;
  output pd_out;
  wire cki,ckq,pd_out;

  detff I1 (rst,dat,ck0,cki);
  detff I2 (rst,dat,ck1,ckq);
  detff_ngt I3 (rst,ckq,cki,pd_out);
endmodule

//フリップフロップ回路
module detff(rst,dat,ck,out);
  input rst,dat,ck;
  output out;
  reg out,cki,ckq;

  always @ (ck or posedge rst) begin
    if(rst==1'b1)
      out <= 1'b0;
    else
      out <= dat;
  end
endmodule

//反転型のフリップ

フロップ回路
module detff_ngt(rst,dat,ck,out);
  input rst,dat,ck;
  output out;
  reg out,tmp_1b;

  always @ (ck or posedge rst) begin
    if(rst==1'b1)
      out <= 1'b0;
    else begin
      tmp_1b = (ck==1'b1)? dat : ~dat;
      out <= tmp_1b;
    end
  end
endmodule
```

リスト6　dgtl_dly モジュール（dgtl_dly.v）

```
//ディジタル遅延回路
//Revised on 15:26 05/09/24,,, Digital Delay
//in, out :: 1bit

module dgtl_dly(in,out,vdd,vss);
  input vdd,vss;
  input in;
  output out;
  reg out;
  reg cnt_strt;              //Counter開始フラグ
  reg [1:50] in_pr;          //1～50clock前のin値

  integer cycle, tmp_i1, tmp_i2;
  integer k_div2;
  integer cnt_tdrf, cntup_trf;
  integer ind;
  real tmp_r1, tmp_r2;

  parameter TM_PRCS=0.001;  //[ns]，時間単位
  parameter TD=0.2;    //[ns]，遅延時間
  parameter TRF=0.2;   //[ns]，立ち上がり・立ち下がり時間
  parameter K_DIV=50;  //(TD+TRF)/K_DIV値を処理サイクル
                                                (cycle)とする

  initial begin
    tmp_i1=K_DIV;
    k_div2=(tmp_i1 > 50)? 50:tmp_i1;//Limitter for K_DIV

    tmp_r1=(TD+TRF)/(k_div2*TM_PRCS);
    tmp_i1=tmp_r1;
    cycle=(tmp_i1 < 1)? 1: tmp_i1;

    tmp_r1=cycle;
    tmp_r2=TRF/(tmp_r1*TM_PRCS);
    tmp_i1=tmp_r2;
    cntup_trf=(tmp_i1 < 1)? 1: tmp_i1;

    cnt_tdrf=0;
    cnt_strt=1'b0;
  end

  always # (cycle) begin

    for(ind=1;ind<k_div2;ind=ind+1) in_pr[ind]
    <= in_pr[ind+1];   //inの遅延
    in_pr[k_div2] <= in;
    out <= in_pr[1];

    if(cnt_strt==1'b0) begin
      cnt_tdrf=0;
      tmp_i1=(in==in_pr[k_div2])? 0:1;
    end else begin
      cnt_tdrf=cnt_tdrf +1;

      if(cnt_tdrf < cntup_trf) begin
        if(in==in_pr[k_div2])
          tmp_i1=1;
        else begin
          tmp_i1=0;
          for(ind=1;ind<=cnt_tdrf;ind=ind+1)
                      //cntup_trf未満の周期を持つパルスを削除
                   in_pr[k_div2-ind] <= in;
        end
      end else
        tmp_i1=0;
    end
    cnt_strt <= tmp_i1;
  end

endmodule
```

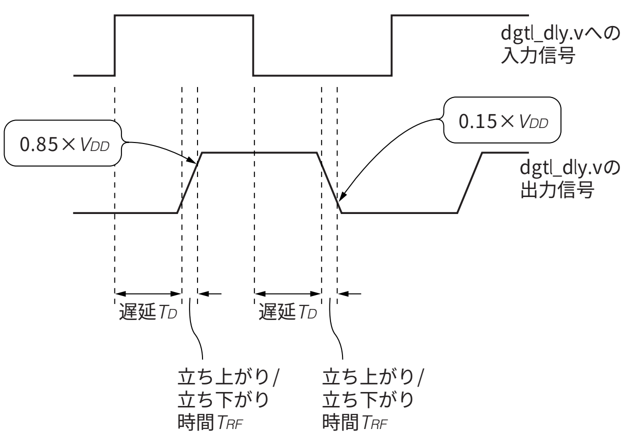


図12　dgtl_dly.v モジュールの遅延 T_D と立ち上がり/立ち下がり時間 T_{RF} の定義

dgtl_dly.v モジュールのパラメータのうち，遅延時間 T_D と立ち上がり/立ち下がり時間 T_{RF} の定義を行う．今回，1ビット期間 0.1ns よりも回路遅延のほうが大きい場合を想定して，T_D と T_{RF} の区別を行った．つまり，T_{RF} より短いパルスは dgtl_dly.v モジュールを通過できない．そのため，図中の T_D と T_{RF} の定義はユニーク（一意的）になっている．

リスト7　cp_lpf モジュール（cp_lpf.v）

```
//チャージ・ポンプ回路＋ローパス・フィルタ回路
//Revised on 01:48 05/11/22
//pd1p, pd1n, fdp, fdn, vout :: 符号付きinteger,
                              4ビット・オーバ＋20ビット・コア

module cp_lpf(rst, pd1p, pd1n, fdp, fdn, vout, vdd,
                                                   vss);
  input rst;
  input pd1p, pd1n, fdp, fdn;
  input vdd, vss;
  output [23:0] vout;
  reg rst_rk;  //ルンゲクッタ計算の初期化フラグ
  reg cal_rk;  //ルンゲクッタ計算の有効化フラグ
  reg good_err=1'b0;
         //ルンゲクッタ計算中でエラーが十分小さいか否かのフラグ
  reg tmp1b_1;
  reg [23:0] vout;
  reg [31:0] thrs1_32b, thrs2_32b, tmp32b_1,
                              tmp32b_lmtH, tmp32b_lmtL;

  integer cycle;
         //ルンゲクッタ処理の実行頻度の逆数
  integer h_n, h_n0;
         //ルンゲクッタ計算でのスキップ数，スキップ数の初期値
  integer cnt_dwn_rk;
         //ルンゲクッタ計算におけるスキップ数のカウンタ
  integer hit_min_h;
         //ルンゲクッタ計算における最小時間刻みの使用頻度
                                          (CP出力更新時を除く)
  integer tmp_i1, tmp_i2;
  real ip_pd1, in_pd1, ip_fd, im_fd, in_fd;
  real i_pd1, i_fd;
  real rl_iin, a11, a12, a21, a22, b1;
  real rl_b1_iin, rl_x1, rl_x2, k1a, k1b, k2a, k2b, k3a,
                                                      k3b;
  real dif_x1, dif_x2, err_x1, err_x2;
  real rl_relerr, rl_abserr;
  real tmp_r1, tmp_r2, tmp_r3, tmp_r4;

  parameter TM_PRCS=0.001;        //[ns]，時間単位
  parameter K_RTOI=524288.0;
         //整数値と内部計算用実数値と変換比率．整数2^20に対して
                                                 実数2.0が対応
  parameter CUR_CP_PD1=5.0;
         //[uA]，位相検出(PD)に関わる充放電電流値，
                                            具体的には正側電流値
  parameter RTO_B_A_PD1=1.0;
         //負側出力電流値の正側出力電流値に対する比(PD)
```

```
parameter CUR_CP_FD=5.0;
    //[uA], 周波数検出(FD)に関わる充放電電流値,
                                   具体的には正側電流値
parameter RTO_B_A_FD=1.0;
    //負側出力電流値の正側出力電流値に対する比(FD)
parameter OFS_CUR_CP_FD=0.0;
    //[uA], 周波数検出(FD)に関わる中間レベル電流値
parameter CP=8.0;          //[pF], LPFのキャパシタ値
parameter C2=0.2;
    //[pF], LPFのキャパシタ値(クロック・フィードスルー抑制用)
parameter RP=1000.0;       //[Ohm], LPFの抵抗値
parameter TS_RK=0.001;     //[ns], ルンゲクッタ処理の周期
parameter N_STP_INI=0.1;
    //[ns], ルンゲクッタ計算のステップ幅の初期値
                                   (LPF出力の更新周期)
parameter RELERR=0.001;   //許容相対エラー項
parameter ABSERR=0.00001; //許容絶対エラー項

initial begin
  rl_relerr=RELERR;
  rl_abserr=ABSERR;
  rst_rk=1'b1;             //サイクル数のリセット
  hit_min_h=0;             //最小時間刻みの使用頻度を0にセット

  tmp_r1=TS_RK/TM_PRCS;
  tmp_i1=tmp_r1;
  cycle=(tmp_i1 < 1)? 1:tmp_i1;
  tmp_r2=cycle;
  a11=-(TM_PRCS*1000.0*tmp_r2)/(C2*RP);
  a12=-a11;
  a21=(TM_PRCS*1000.0*tmp_r2)/(CP*RP);
  a22=-a21;
  b1=(TM_PRCS*1000.0*tmp_r2)/C2;

  tmp_r1=cycle;
  tmp_r2=(N_STP_INI/(TM_PRCS*cycle));
  tmp_i1=tmp_r2;
  h_n0=(tmp_i1 < 1)? 1:tmp_i1;

  ip_pd1=CUR_CP_PD1*0.000001;
   //[A], PDにおける正側出力電流値
  in_pd1=-RTO_B_A_PD1*CUR_CP_PD1*0.000001;
   //[A], PDにおける負側出力電流値
  ip_fd=CUR_CP_FD*0.000001;
   //[A], FDにおける正側出力電流値
  in_fd=-RTO_B_A_FD*CUR_CP_FD*0.000001;
   //[A], FDにおける負側出力電流値
  im_fd=OFS_CUR_CP_FD*0.000001;
   //[A], FDにおける中間レベル出力電流値(理想はゼロ)
  rl_x1=0;
  rl_x2=0;

  tmp32b_lmtH={8'h00, 24'h7F_FFFF};
  tmp32b_lmtL=~tmp32b_lmtH;
end

//チャージ・ポンプ
always @ (pd1p or fdp) begin
  if(pd1p==1'b1) i_pd1=ip_pd1;
  else i_pd1=in_pd1;

  if((fdp==1'b1) &(fdn==1'b0)) i_fd=ip_fd;
  else if((fdp==1'b0) &(fdn==1'b1)) i_fd=in_fd;
  else i_fd=im_fd;

  //ルンゲクッタ用
  rl_iin=i_pd1-i_fd;
  rl_b1_iin=b1*rl_iin;
  rst_rk <= 1'b1;
end

//Low Pass Filter
always #(cycle) begin
  if(rst==1'b1) begin
    rl_x1=0;
    rl_x2=0;

    //ルンゲクッタ用
    rl_b1_iin=0.0;
    rst_rk <= 1'b1;
    h_n=h_n0;
  end else begin
  //ルンゲクッタ：2次と3次の埋め込み式
    if(rst_rk==1'b1) begin
      tmp_i1=h_n-cnt_dwn_rk;
      tmp_i2=(tmp_i1 < 1)? 1: tmp_i1;
      tmp_r1=h_n;
      tmp_r2=tmp_i2;
      tmp_r3=tmp_r2/tmp_r1;
      rl_x1 = rl_x1 +tmp_r3*k2a;  //状態値の更新
      rl_x2 = rl_x2 +tmp_r3*k2b;  //状態値の更新

      h_n=h_n0;
      cnt_dwn_rk=h_n0;
      cal_rk=1'b1;
    end else begin
      if(cnt_dwn_rk==0) begin
        rl_x1 = rl_x1 +k2a;       //状態値の更新
        rl_x2 = rl_x2 +k2b;       //状態値の更新

        h_n = (h_n << 2);
        cnt_dwn_rk=h_n;
        cal_rk=1'b1;
      end else
        cal_rk=1'b0;
    end

    if(cal_rk==1'b1) begin
      while (cal_rk==1'b1) begin
      //ルンゲクッタ計算のコア部
        tmp_r3=h_n;
        k1a=(a11*rl_x1 +a12*rl_x2 +rl_b1_iin)*tmp_r3;
        k1b=(a21*rl_x1 +a22*rl_x2)*tmp_r3;

        tmp_r1=rl_x1 +k1a*0.5;
        tmp_r2=rl_x2 +k1b*0.5;
        k2a=(a11*tmp_r1 +a12*tmp_r2
                                  +rl_b1_iin)*tmp_r3;
        k2b=(a21*tmp_r1 +a22*tmp_r2)*tmp_r3;

        tmp_r1=rl_x1 -k1a +k2a*2;
        tmp_r2=rl_x2 -k1b +k2b*2;
        k3a=(a11*tmp_r1 +a12*tmp_r2
                                  +rl_b1_iin)*tmp_r3;
        k3b=(a21*tmp_r1 +a22*tmp_r2)*tmp_r3;

        dif_x1=(-k1a +k2a*2.0 -k3a)/6.0;
        dif_x2=(-k1b +k2b*2.0 -k3b)/6.0;
        tmp_r3 = rl_x1 +(k1a +k2a*4 +k3a)/6.0;
        tmp_r4 = rl_x2 +(k1b +k2b*4 +k3b)/6.0;

        tmp_r1=(tmp_r3 > 0)? tmp_r3:-tmp_r3;
        tmp_r2=(tmp_r4 > 0)? tmp_r4:-tmp_r4;
        err_x1=tmp_r1*rl_relerr + rl_abserr;
        err_x2=tmp_r2*rl_relerr + rl_abserr;
        tmp_r1=(dif_x1 > 0)? dif_x1:-dif_x1;
        tmp_r2=(dif_x2 > 0)? dif_x2:-dif_x2;

        good_err=((err_x1 >= tmp_r1) && (err_x2 >=
                                 tmp_r2))? 1'b1:1'b0;
        if((good_err==1'b1) || (h_n==1)) begin
          rst_rk <= 1'b0; //初期化を解除
          cal_rk = 1'b0;  //ルンゲクッタ計算ループから出る
        end else begin
          tmp_i1 = h_n >> 2;
          h_n=(tmp_i1 < 1)? 1: tmp_i1;
          hit_min_h=(tmp_i1 < 1)? (hit_min_h+1) :
                                            hit_min_h;
          cnt_dwn_rk=h_n;
        end
      end
    end
    cnt_dwn_rk=cnt_dwn_rk-1;//ダウンカウンタのデクリメント
  end

  tmp_r3=rl_x1*K_RTOI;
  tmp32b_1=$rtoi(tmp_r3);
  if(tmp32b_1[31]==1'b0)
    vout <= (tmp32b_1 > tmp32b_lmtH)?
      tmp32b_lmtH[23:0] : tmp32b_1[23:0];   //リミッタ
  else
    vout <= (tmp32b_1 < tmp32b_lmtL)?
      tmp32b_lmtL[23:0] : tmp32b_1 [23:0];  //リミッタ
end

endmodule
```

図10ではpfd_topモジュール(pdf_top.v)の構成要素として1個のdgtl_dlyモジュール(dgtl_dly.v)を示しましたが，実際には**図4**における差動出力であるV_{PD}とV_{FD}に対応した4個のdgtl_dlyモジュールが存在します．**リスト4**(p.122)がソース・コードです．topモジュールから受け取ったパラメータ値は，dgtl_dlyモジュールへと引き渡されます．

4) pfdモジュール(pfd.v)

pfdモジュールは**図4**を実現しています．**リスト5**(p.123)がソース・コードです．**図4**以外の端子として，**リスト5**では，リセット信号(`rst`)と電源(`vdd`，`vss`)を追加しています．

5) dgtl_dlyモジュール(dgtl_dly.v)

dgtl_dlyモジュールは，**図4**の入力信号(入力データや生成クロック)から差動出力であるV_{PD}，V_{FD}までの遅延を実現しています．**リスト6**(p.124)がソース・コードです．四つのパラメータ値をpfd_topモジュールから受け取ります．`TM_PRCS`はシミュレーション時間単位(`` `timescale ``で与えたのと同じ値)，`TD`や`TRF`は遅延(定義は**図12**(p.124)のとおり．`TRF`はパルス幅の短いパルスを消滅させるためのパラメータ)，`K_DIV`は処理時間の間隔です(`(TD+TRF)/K_DIV`が処理時間の間隔を決める．なお，実行時間の短縮と遅延処理の精度はトレードオフの関係にある．また，`K_DIV`の上限値を50とした)．

6) cp_lpfモジュール(cp_lpf.v)

図6および**図7**を実現します．**リスト7**がソース・コードです．14個のパラメータ値をtopモジュールから受け取

COLUMN 《時間刻み幅可変の精度2と精度3のルンゲクッタ法による埋め込み型アルゴリズム》

常微分方程式の初期値問題(初期値が与えられた場合の常微分方程式を解く問題)として，ルンゲクッタ法があります．ルンゲクッタ法はオイラー法(精度1)から派生したもので，精度8までのアルゴリズムが紹介されています(2)．なお，ここで精度とは，演算結果と真値のエラー(最大エラー)が時間刻み幅の何乗に比例するかを意味しています．それゆえ，演算エラーについて，精度1であれば時間刻み幅に比例し，精度8であれば時間刻み幅の8乗に比例します．精度が高いほうが時間刻み幅を大きくできますが，1回当たりの演算量が増えます．

今回，計算精度の維持とシミュレーション時間の短縮という目的において，精度2と精度3のルンゲクッタ法の演算結果の比較により，時間刻み幅を可変にする解法(アルゴリズム)を用います．詳細な根拠の解説は参考文献(2)に譲りますが，ひととおり説明します．

まず，本文の**図7**を状態方程式で表します．**図7**に示すように状態変数をx_1とx_2にすれば，状態方程式は式(B-1)になります．なお，x_1がそのまま出力電圧になります．

$$\frac{dX}{dt}\begin{bmatrix} -\dfrac{1}{C2*Rp} & \dfrac{1}{C2*Rp} \\ \dfrac{1}{Cp*Rp} & -\dfrac{1}{Cp*Rp} \end{bmatrix} * X + \begin{bmatrix} \dfrac{1}{C2} \\ 0 \end{bmatrix} * Iin \qquad \cdots\cdots\text{(B-1)}$$

$$X=\begin{bmatrix} x1 \\ x2 \end{bmatrix}$$

次に，時間刻み幅をhとすれば，精度2のルンゲクッタ法の推測値X_{i+1}，精度3のルンゲクッタ法の推測値$\tilde{X}_{i+1}$，および精度2と精度3の推測値の差は，式(B-1)の結果を受けて式(B-2)ようになります．

$$X_{i+1}=X_i+h*k2$$

$$\tilde{X}_{i+1}=X_i+h*\left\{\frac{1}{6}*k1+\frac{2}{3}*k2+\frac{1}{6}*k3\right\}$$

$$X_{i+1}-\tilde{X}_{i+1}=h*\left\{-\frac{1}{6}*k1+\frac{1}{3}*k2-\frac{1}{6}*k3\right\}$$

$$k1=\left.\frac{dX}{dt}\right|_{X=X_i} \qquad \cdots\cdots\text{(B-2)}$$

$$k2=\left.\frac{dX}{dt}\right|_{X=X_i+0.5*h*k1}$$

$$k3=\left.\frac{dX}{dt}\right|_{X=X_i-h*k1+2*h*k3}$$

さらに，上式(B-2)について，与えられた精度を満足するように時間刻み幅を変更しながら，演算を実行します．すなわち次の手順(STEP1～3)に従います．ただし，本文の**図7**の入力I_{in}の更新時は時間刻み幅を最小幅へと無条件にセットします．これは，推測エラーが大きくなることが予想されるからです．

- STEP1：$t=t_i$のときのX_iをもとに，$t=t_i+h$のときのX_{i+1}と$\tilde{X}_{i+1}$を求める
- STEP2：次式(B-3)で定義する目標エラーerrを計算する．ただし，$RELERR$と$ABSERR$は，それぞれ許容相対エラー，許容絶対エラーである．

$$err=\left\|\tilde{X}_i\right\|*RELERR+ABSERR \qquad \cdots\cdots\text{(B-3)}$$

- STEP3：上式(B-3)の結果errと$\| X_{i+1}-\tilde{X}_{i+1} \|$を比較して，時間刻み幅を更新する．つまり，以下のとおりである．

  $\| X_{i+1}-\tilde{X}_{i+1} \| \leqq err$のとき，$t=t_i+1$のときのX値としてSTEP1で求めた$X_{i+1}$で良しとする．時間刻み幅を$h\rightarrow 4\times h$に変更したうえでSTEP1を実行する($X_{i+2}$を求める)

  $\| X_{i+1}-\tilde{X}_{i+1} \| > err$のとき，時間刻み幅を$h\rightarrow 0.25\times h$に変更したうえでSTEP1に戻って再度$X_{i+1}$を求める．ただし，時間刻み幅の変更値が最小刻み幅となった場合は，STEP1で求めたX_{i+1}で良しとする．そのうえで，時間刻み幅はそのままでSTEP1を実行する(X_{i+2}を求める)．

ります．TM_PRCSはシミュレーション時間単位（`timescaleで与えたのと同じ値），K_RTOIは内部変数（実数）と信号（整数表現）の比（係数）です．CUR_CP_PD1，CUR_CP_FD，RTO_B_A_PD1，RTO_B_A_FD，OFS_CUR_CP_FDは，PD用もしくはFD用で，素子のミスマッチやオフセットによって（**図6**），正側電流や負側電流，もしくはその中間レベル電流の間で比例関係が崩れた場合を想定したパラメータです．それぞれの出力電流の算出式は，**リスト7**のinitialブロック中のip_pd1，in_pd1，ip_fd，in_fd，im_fdの算出箇所に相当します．また，CP，C2，RPは**図7**のとおりです．TS_RK，N_STP_INI，RELERR，ABSERRは**図7**のフィルタ出力（常微分方程式）を数値計算で解く際のパラメータです（左掲のコラム「時間刻み幅可変の精度2と精度3のルンゲクッタ法による埋め込み型アルゴリズム」を参照）．なお，**図6**および**図7**以外の端子として，**リスト7**では，リセット信号（rst）と電源（vdd，vss）を追加しています．

7）re_timeモジュール（re_time.v）

re_timeモジュールは，**図10**のようにvco_chnモジュールで生成したクロックによって入力データを取り込みます．具体的な回路は，**図4（a）**のPD回路に使用している差動型フリップフロップ回路と同じになります．また，**リスト8**がソース・コードです．

● 入力データ生成回路：23ビットの疑似乱数を発生

図10におけるdat_gen.vは，23ビット（$2^{23}-1$の乱数）の疑似乱数生成回路です．疑似乱数はM系列によって生成しています（p.119のコラム「M系列による疑似乱数の生成」を参照）．**リスト9**がソース・コードです．

● 測定回路：リファレンス信号と位相を比較

図10におけるmsjt.vは測定回路です．**リスト10**がソース・コードです．strt信号が“H”で測定状態となり，入力データ列を作るリファレンス信号ref_clkとvco_chnモジュールで内部生成したクロックgen_clk_m90の位相を比較することで（理想的には，ref_clkとgen_clk_m90は90°位相が異なる），ジッタ量を測定します（**図9**）．また，5個のパラメータ値をtopモジュールから受け取ります．TM_PRCSはシミュレーション時間単位（`timescaleで与えたのと同じ値），CYCLE_MJは測定を行う際のスキップ数，T_AVRGはジッタ量の平均値を求めるためのサンプル期間，T_AVMJはピーク・ジッタ量やRMSジッタ量を求めるためのサンプル期間，TS_CLKは入力データを生成するためのリファレンス・クロック周期です．

リスト8　re_timeモジュール（re_time.v）

```
//データ再生成回路
//Revised on 18:40 05/08/11
//rst,ckp,ckn,dat_inp,dat_inn,dat_outp,dat_outn ::
                                                1bit

module re_time(rst,ckp,ckn,dat_inp,dat_inn,dat_outp,
                                   dat_outn,vdd,vss);
  input rst,ckp,ckn,dat_inp,dat_inn;
  input vdd,vss;
  output dat_outp,dat_outn;
  reg dat_out;
  wire dat_outp,dat_outn,ck,dat_in;

  assign dat_outp=(dat_out==1'b0)? 1'b0 : 1'b1;
  assign dat_outn=~dat_outp;
  assign ck=(ckp==~ckn)? ckp : 1'b0;
  assign dat_in=(dat_inp==~dat_inn)?
               dat_inp : dat_inn;

  always @ (posedge ck or negedge ck) begin
    if(rst==1'b1)
      dat_out <= 1'b0;
    else
      dat_out <= dat_in;
  end

endmodule
```

リスト9　疑似乱数生成回路（dat_gen.v）

```
//入力データ生成回路
//Revised on 11:37 05/08/20,, Data Generator,
//23bit-M-series
//pfd_outp,pfd_outn :: signed-integer,
//4bit-over + 20bit-core

module dat_gen(rst,clk,datp,datn);
  input rst,clk;
  output datp,datn;
  reg x_prbs[1:23];
  wire datp,datn,dat;

  assign dat=x_prbs[23];
  assign datp=dat;
  assign datn=~dat;

  integer index;

  always @ (clk or posedge rst) begin
    if(rst==1'b1) begin
      for(index=1;index<24;index=index+1)
        x_prbs[index] <= 1'b1;
    end else begin
      for(index=1;index<23;index=index+1) begin
        x_prbs[24-index] <= x_prbs[23-index];
      end
      x_prbs[1] <= (x_prbs[18] ^ x_prbs[23]);
    end
  end

endmodule
```

リスト10　測定回路(msjt.v)

```
//測定回路
//Revised on 14:19 05/09/03
//Evaluate peak to peak, rms jitter between ref-clock
and 135deg(45deg+90deg)-vco-genclk
//But we assume that peak jitter is within +/- 50psec(-
90deg~+90deg)

module msjt(strt,ref_clk,gen_clk_m90);
  input strt,ref_clk,gen_clk_m90; //gen_clk_m90 => vout45
  reg ref_clk_pr,gen_clk_m90pr;
  reg en_mj_tsh;

  integer cycle;
  integer ts_av,cnt_av,ts_avmj,tm_avmj,ts_90deg,ts_180deg;
  integer mj_err;
  integer flg_set_gen_clk_m90;
     //VCO生成クロックのエッジで1となり，リファレンス・クロック
                                                  のエッジで0となる
  integer flg_set_gen_clk_m90tsh;
     //VCO生成クロックのエッジで1となり，それ以外のVCO状態で
                                                          0となる
  integer cnt_try_av;
  integer mj_cnt,mj_avr,rslt_av;
  integer tsh_min_gen_clk,tsh_max_gen_clk;
  integer cnt_tsh;
  integer mj_pk_min,mj_pk_max;
  integer tmp_i1,tmp_i2;
  integer hit_jtrms,mj_rms2_i;
  real tmp_r1,tmp_r2,tmp_r3;

  parameter TM_PRCS=0.001; //[ns]，時間単位
  parameter CYCLE_MJ=1;
     //測定を行う際のスキップ数．1/CYCLE_MJが測定頻度を意味する
  parameter T_AVRG=500;
     //[ns]，strtがL->H後のジッタ平均値を求めるための時間を指す
  parameter T_AVMJ=3000;
     //[ns]，ピークジッタやRMSジッタ値を求めるための時間を指す
  parameter TS_CLK=0.2;  //[ns]，リファレンス・クロックの周期

  initial begin
    gen_clk_m90pr=1'b0;  //1 clock前のVCO Clock(vout45)
    ref_clk_pr=1'b0;     //1 clock前のRef Clock
    en_mj_tsh=1'b0;      //VCOの半周期測定フラグをOFFに設定
                         (strt="H"時にONとする)
    cycle=CYCLE_MJ;

    tmp_r1=T_AVRG/(TM_PRCS*cycle);
    ts_av=tmp_r1; //発生ジッタ量の平均値を求める期間(int)
    tmp_r1=T_AVMJ/(TM_PRCS*cycle);
    ts_avmj=tmp_r1; //ピーク・ジッタやRMSジッタ量を
                                               測定する期間(int)
    tm_avmj=ts_avmj+ts_av; //rms & peak jitterの
                                               算出終了時刻(int)

    flg_set_gen_clk_m90=0;
    flg_set_gen_clk_m90tsh=0;

    tmp_r1=TS_CLK*0.25/(TM_PRCS*cycle);
    ts_90deg=tmp_r1;       //90degに相当する期間(int)
    ts_180deg=ts_90deg*2;  //180degに相当する期間(int)

    cnt_try_av=0; //VCO生成クロックとリファレンス・クロックの
                    エッジが180deg期間でミートした回数をカウント
    tsh_min_gen_clk=ts_180deg; //VCO周期の最小値(int)
    tsh_max_gen_clk=ts_180deg; //VCO周期の最大値(int)
    cnt_av=0;    //観測用のカウンタ(#cycleごとに更新)
    cnt_tsh=0;   //VCO-Clockエッジからの経過時間を対象とした
                                                         カウンタ
    mj_cnt=0;    //jitterを求めるためのカウンタ
    mj_avr=0;    //jitterの平均を求めるための総和(int)
    mj_rms2_i=0; //jitterのrms^2を求めるための変数(int)
    mj_pk_min=0; //jitterのpeak最小値を求めるための変数(int)
    mj_pk_max=0; //jitterのpeak最大値を求めるための変数(int)
    hit_jtrms=0; //jitterのrms^2計算に使用したサンプル数
                                                            (int)
  end

  always # (cycle) begin
    if(strt==1'b0)  cnt_av=0;
    else begin
      ref_clk_pr <= ref_clk;
      gen_clk_m90pr <= gen_clk_m90;
      if(cnt_av < ts_av) begin  //発生ジッタ量の平均値を測定
        tmp_i1=flg_set_gen_clk_m90;
        if(tmp_i1==0) begin
          mj_cnt=0;
          flg_set_gen_clk_m90=(gen_clk_m90
                                    ==gen_clk_m90pr)? 0:1;
        end else begin
          mj_cnt=mj_cnt+1;
          flg_set_gen_clk_m90=(ref_clk==ref_clk_pr)?
                                                        1:0;
          if((ref_clk != ref_clk_pr) && (mj_cnt <=
                                         ts_180deg)) begin
            mj_avr=mj_avr +(mj_cnt-ts_90deg);
            cnt_try_av=cnt_try_av+1;
          end
        end
        cnt_av=cnt_av+1;

      end else if(cnt_av == ts_av) begin
                                  //発生ジッタ量の平均値を計算
        if(cnt_try_av==0) begin
          $display("Did not lock... So quit now.");
          $stop;
        end else begin
          tmp_r1=mj_avr;
          tmp_r2=cnt_try_av;
          tmp_r3=tmp_r1/tmp_r2;
          rslt_av=tmp_r3;
        end
        cnt_av=cnt_av+1;

      end else if(cnt_av > tm_avmj)         //測定しない
        cnt_av=tm_avmj+2;

      else begin  //"cnt_av > ts_av" && "cnt_av <=
        tm_avmj"を満たすケース発生ジッタのピーク値やRMS値を測定
                   //VCOで生成したクロックの最小半周期と最大
                                                   半周期を測定
        tmp_i1=flg_set_gen_clk_m90;
        if(tmp_i1==0) begin
          mj_cnt=0;
          flg_set_gen_clk_m90=(gen_clk_m90
                                    ==gen_clk_m90pr)? 0:1;
        end else begin
          mj_cnt=mj_cnt+1;
          flg_set_gen_clk_m90=(ref_clk==ref_clk_pr)?
                                                        1:0;
          if((ref_clk != ref_clk_pr) && (mj_cnt <=
                                         ts_180deg)) begin
            mj_err=mj_cnt -ts_90deg -rslt_av;
            mj_rms2_i=mj_rms2_i +(mj_err*mj_err);
            mj_pk_min=(mj_err < mj_pk_min)? mj_err :
                                                  mj_pk_min;
            mj_pk_max=(mj_err > mj_pk_max)? mj_err :
                                                  mj_pk_max;

            hit_jtrms=hit_jtrms+1;
          end
        end

        flg_set_gen_clk_m90tsh=(gen_clk_m90
                                    ==gen_clk_m90pr)? 0:1;
        if(flg_set_gen_clk_m90tsh==0)
          cnt_tsh=(en_mj_tsh==1'b1)? (cnt_tsh+1) : 1;
                   //VCOで生成したクロック周期をカウント
        else begin
          if(en_mj_tsh==1'b1) begin
                   //VCOで生成したクロック周期の最小と最大を更新
            tsh_min_gen_clk=(cnt_tsh < tsh_min_gen_clk)?
                                cnt_tsh : tsh_min_gen_clk;
            tsh_max_gen_clk=(cnt_tsh > tsh_max_gen_clk)?
                                cnt_tsh : tsh_max_gen_clk;
          end else
            en_mj_tsh=1'b1;
          cnt_tsh=1;
        end
        cnt_av=cnt_av+1;
      end
    end
  end

endmodule
```

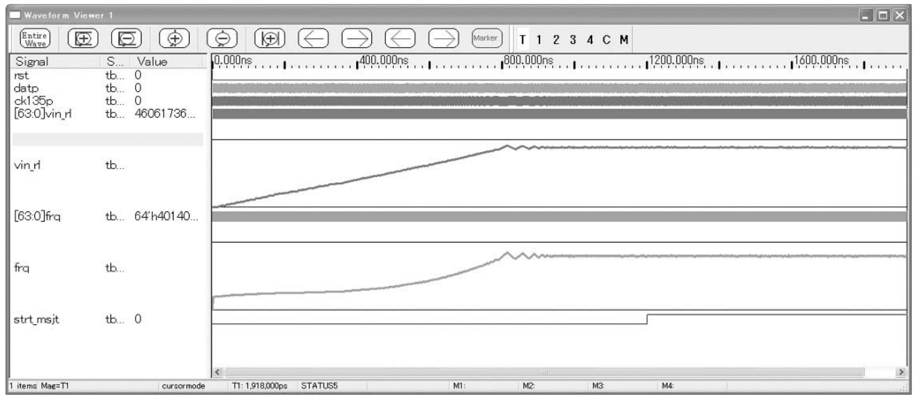


図13
リスト1によるシミュレーション結果（0～1.9μs）
vin_rl（VCO入力電圧）とfrq（VCO発振周波数）が，1.0μs付近でロックしてることがわかる．

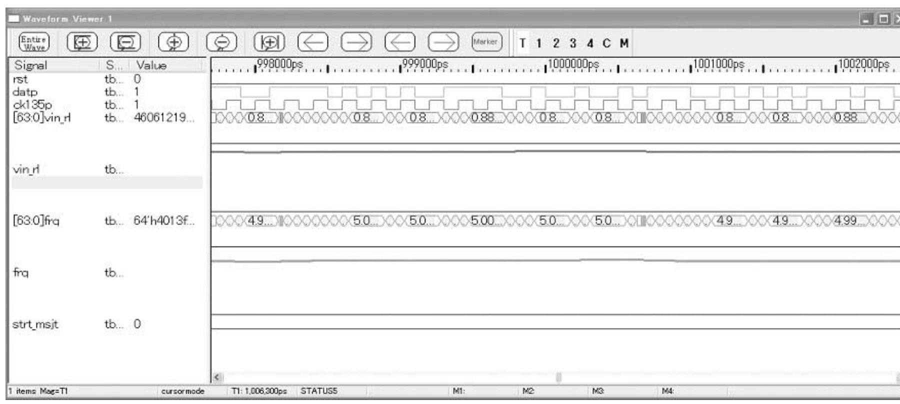


図14
リスト1によるシミュレーション結果（1.0 μs付近）
datpとck135pが同期しているのを確認できる．

4 シミュレーション結果

今回，デバイス・ノイズは考慮していませんが，ディジタル遅延値やローパス・フィルタの定数，チャージ・ポンプの電流値の調整しだいで，ジッタ量やロックするまでの時間が変わります．ここでは，ディジタル遅延値に着目して，ジッタ量やロックするまでの時間を確認します．なお，シミュレーション時間については，2.4GHz動作のCeleron，736Mバイトのメイン・メモリ，Windows XP（Home Edition，SP2）で**リスト1**のテストベンチをVeritakWinで実行したところ，7～8分で処理が終了しました．

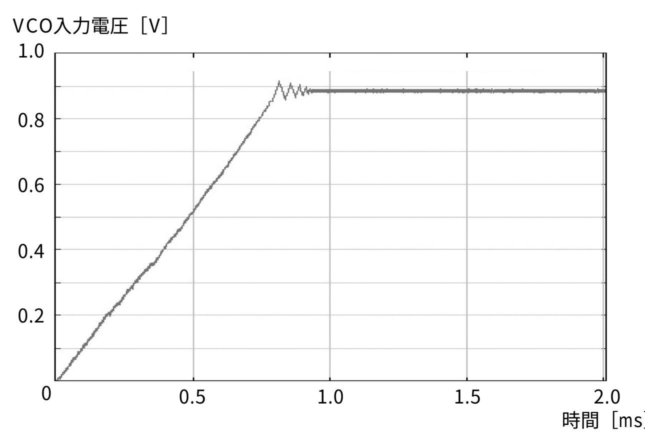


図15　リスト1のパラメータ値でのシミュレーション結果
リスト1（テストベンチ）のパラメータ値を用いたシミュレーション結果．VCO入力電圧がロックするまでの時間（ロック時間）やロック状態におけるリプルを確認できる．

● シミュレーションして結果を確認

リスト1をVeritakWinでシミュレーションしました．`rst`（リセット信号），`datp`（入力データ），`ck135p`（VCOで内部生成したクロック），`vin_rl`（VCO入力電圧［V］，実数値），`frq`（VCO発振周波数［GHz］，実数値），および`strt_msjt`（測定開始フラグ）に着目してプロットしたのが**図13**と**図14**です．**図13**は0～1.9μs期間のプロットです．`vin_rl`と`frq`はアナログ波形表示で，1.0μs付近ですでにロックしていることがわかります．また，**図14**は1.0μs時あたりを拡大したもので，`datp`と`ck135p`が同期していることを確認できます．

図15は，外部ファイルrslt.datへ書き込んだ結果です．

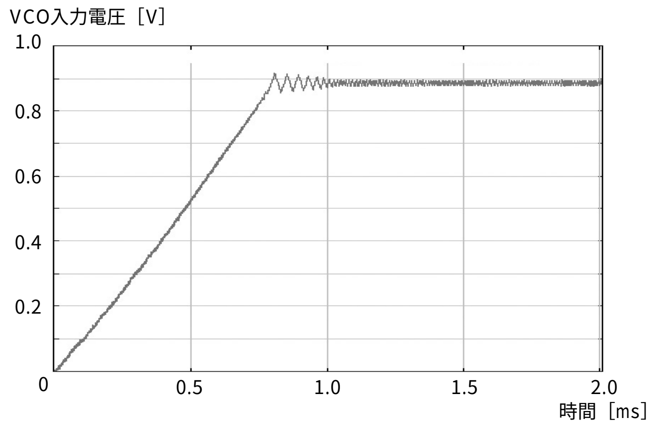


図16　ディジタル遅延値（TD_PD1とTD_FD）を2倍にした場合のシミュレーション結果

VCO入力電圧（**図13**の`vin_rl`に相当）の時間応答をgnuplotでプロットしました．さらに，rslt.datファイルの最後尾に測定したジッタ量が書き込まれます（**表1**を参照．ただし，単位はRMSの2乗値がps^2で，これと許容誤差を満足しない回数を除いた結果の単位はps）．ジッタ量のピーク値とRMS値を**表2**に示します．目標としたジッタ量をクリアしています．

表1　測定したジッタ量

ジッタ量	平均値	－2
	正側ピーク値	2
	負側ピーク値	－1
	RMSの2乗値	0.578716
VCO生成クロック半周期	最小値	99
	最大値	101
ルンゲクッタ計算で許容誤差を満足しなかった回数	－	0

表2　シミュレーション結果（ジッタ量）

ディジタル遅延値（TD_PD1とTD_FD）	ピーク［ps］	RMS［ps］	備　考
TD_PD1＝0.3ns TD_FD＝0.5ns	3	0.761	リスト1のパラメータ値を採用．図15も参照
TD_PD1＝0.6ns TD_FD＝1.0ns	6	1.421	左記のディジタル遅延値以外は，リスト1のパラメータ値を採用． 図16も参照

● ディジタル遅延値を変更してシミュレーション

リスト1で指定したパラメータ値のうち，ディジタル遅延に関する`TD_PD1`と`TD_FD`の二つのパラメータを，それぞれ0.6と1.0に変更し，シミュレーションを実行しました．

上記と同じように，VCO入力電圧の応答を**図16**に示します．**図15**と比べて，ロックした状態のリプルが大きいことを確認できます．また，ジッタ量のピーク値とRMS値を**表2**に示します．上記と比べてジッタ量は大きくなり，RMS値で目標をクリアできていません．

今回の例では煩雑さを避けるため，デバイス・ノイズを無視しました．これを考慮する場合には，あらかじめノイズ信号を外部ファイルとして用意して，感度の高いVCO入力に`$readmemh`コマンドなどで加える方法があります．

また，シミュレーション時間を短縮するために，時間単位に1psを指定しました．今，**リスト1**のテストベンチにて，時間単位を小さく，例えば100fsにすると，RMS値（ジッタ量）は0.5ps程度に下がり，ロックに入る付近の大きな振動がほぼなくなります（**図15**を参照．厳密にはVCOのデバイス・ノイズなどの影響も考慮する必要がある）．ちなみに，時間単位を100fsとした場合のテストベンチ（モジュール名はtb_top100fs）を**リスト11**に示します．シミュレーション時間は7～8倍になります．興味があれば，シミュレーションしてみていただきたいと思います．

参考・引用*文献

(1) 日本アルテラ；Mercury 高機能 CDR サポート，http://www.altera.co.jp/products/devices/mercury/features/mcy-cdr.html
(2) Jafar Savoj, Behzad Razavi ; A 10-Gbps CMOS Clock and Data Recovery Circuit With a Half-Rate inary Phase/Frequency Detector, IEEE J. Solid State Circuits, vol.38, pp.13-21, Jan. 2003.
(3) Gisela Engelu-Mullges, Frank Uhlig ; Numerical Algorithms with C, pp.423-488, Springer, 1996.
(4) 藤井裕子；M系列信号の性質と応用，http://www.cqpub.co.jp/DWM/contents/0007/dwm000701150.pdf

ありはら・えいいち
アナログ/ミックスト・シグナルIC コンサルタント

<筆者プロフィール>
在原栄一．アナログ/ミックスト・シグナルIC 設計や機能検証に関する受託設計や技術コンサルティングに従事している．

リスト11　時間単位を100fsとした場合のテストベンチ(tb_top100fs.v)

```
//CDR回路用テストベンチ
//Revised on 01:29 05/11/23

`timescale 100fs / 100fs //<== 時間単位100fsの場合に変更！

`include "top.v"
`include "dat_gen.v"
`include "msjt.v"

module tb_top100fs;
  reg rst, ref_clk;
  reg strt_msjt, ckgen;
  reg msjt_clk;
  reg msjt_clk2;
  wire out_p, out_n, ckgen45_p, ckgen45_n, dat_p, dat_n;
  wire vdd, vss;
  wire [23:0] clf_out;

  integer ts_refclk;
  integer mcd;
  integer tmp_i1,tmp_i2;
  real tmp_r1,tmp_r2;

  //トップ回路(top)と測定回路(msjt)用パラメータ
  parameter TM_PRCS=0.0001;
     //[ns],時間単位 <== 時間単位100fsの場合に変更！

  //トップ回路(top)用パラメータ
  parameter K_RTOI=524288.0; //real変数からinteger変数へ変換係数
  parameter TD_PD1=0.3; //[ns], 遅延時間,PD用
  parameter TRF_PD1=0.1; //[ns], 立ち上がり・立ち下がり時間,PD用
  parameter K_DIV_PD1=20;
              //(TD+TRF)/K_DIV値を処理サイクル(cycle)とする.PD用
  parameter TD_FD=0.5; //[ns], 遅延時間,FD用
  parameter TRF_FD=0.1;  //[ns], 立ち上がり・立ち下がり時間,FD用
  parameter K_DIV_FD=20;
              //(TD+TRF)/K_DIV値を処理サイクル(cycle)とする.FD用
  parameter CUR_CP_PD1=125.0; //[uA],チャージ・ポンプの電流値,
                                                           PD1用
  parameter RTO_B_A_PD1=1.0; //チャージ・ポンプの吸い込み電流
                     と吐き出し電流の比率(Un-matchの影響度),PD用
  parameter CUR_CP_FD=125.0; //[uA],チャージ・ポンプの電流値,FD用
  parameter RTO_B_A_FD=1.0; //チャージ・ポンプの吸い込み電流
                     と吐き出し電流の比率(Un-matchの影響度),FD用
  parameter OFS_CUR_CP_FD=0.0;
              //[uA],中間レベル入力のチャージ・ポンプの電流値,FD用
  parameter CP=50.0; //[pF],LPFでのキャパシタ値(フィルタ定数)
  parameter C2=2.0; //[pF],チャージ・ポンプ切り替り時の
                        フィードスルー緩和用キャパシタ値.LPFで使用
  parameter RP=30.0; //[Ohm],LPFでの抵抗値(フィルタ定数)
  parameter TS_RK_LPF=0.001; //[ns],ルンゲクッタ処理の周期,LPF用
  parameter N_STP_INI_LPF=0.064; //[ns],ルンゲクッタ計算の
                      ステップ幅の初期値(LPF出力の更新周期の初期値)
  parameter RELERR_LPF=0.001; //許容相対エラー項(ルンゲクッタ計算)
  parameter ABSERR_LPF=0.00001; //許容絶対エラー項(ルンゲクッタ計算)
  parameter CYCLE_VCO=1; //VCOでの出力更新時間(=TM_PRCS*CYCLE_VCO)
  parameter THETA_INIT=0.0; //[rad],VCOでの初期位相

  //測定回路(msjt)用パラメータ
  parameter CYCLE_MJ=1;
     //測定を行う際のスキップ数.1/CYCLE_MJが測定頻度を意味する
  parameter T_AVRG=300;
     //[ns],strt_msjtがL->H後のジッタ平均値を求めるための時間を指す
  parameter T_AVMJ=500;
     //[ns],ピーク・ジッタやRMSジッタ値を求めるための時間を指す
  parameter TS_CLK=0.2;
     //[ns],リファレンス・クロックの周期

  defparam I2.TM_PRCS=TM_PRCS;
  defparam I2.K_RTOI=K_RTOI;
  defparam I2.TD_PD1=TD_PD1;
  defparam I2.TRF_PD1=TRF_PD1;
  defparam I2.K_DIV_PD1=K_DIV_PD1;
  defparam I2.TD_FD=TD_FD;
  defparam I2.TRF_FD=TRF_FD;
  defparam I2.K_DIV_FD=K_DIV_FD;
  defparam I2.N_STP_INI_LPF=N_STP_INI_LPF;
  defparam I2.RELERR_LPF=RELERR_LPF;
  defparam I2.ABSERR_LPF=ABSERR_LPF;
  defparam I2.TS_RK_LPF=TS_RK_LPF;
  defparam I2.CUR_CP_PD1=CUR_CP_PD1;
  defparam I2.RTO_B_A_PD1=RTO_B_A_PD1;
  defparam I2.CUR_CP_FD=CUR_CP_FD;
  defparam I2.RTO_B_A_FD=RTO_B_A_FD;
  defparam I2.OFS_CUR_CP_FD=OFS_CUR_CP_FD;
  defparam I2.CP=CP;
  defparam I2.C2=C2;
  defparam I2.RP=RP;
  defparam I2.CYCLE_VCO=CYCLE_VCO;
  defparam I2.THETA_INIT=THETA_INIT;

  defparam I3.TM_PRCS=TM_PRCS;
  defparam I3.CYCLE_MJ=CYCLE_MJ;
  defparam I3.T_AVRG=T_AVRG;
  defparam I3.T_AVMJ=T_AVMJ;
  defparam I3.TS_CLK=TS_CLK;

  //---ネットリスト
  assign vdd=1'b1;
  assign vss=1'b0;
  assign ckgen45=(ckgen45_p==ckgen45_n)? 1'b0 :
                                                  ckgen45_p;
  dat_gen I1 (rst,ref_clk,dat_p,dat_n);
  top I2 (rst, dat_p, dat_n, out_p, out_n, ckgen45_p,
                      ckgen45_n, clf_out, vdd, vss);
  msjt I3 (strt_msjt, ref_clk, ckgen45);

  initial begin
    tmp_r1=TS_CLK/TM_PRCS;
    ts_refclk=tmp_r1;
      //入力データ生成用リファレンス・クロック周期
    mcd=$fopen("rslt.dat");
                       //結果書き込み用ファイルの指定
    rst=1'b1;          //リセット有効
    strt_msjt=1'b0;    //測定無効

    #1000 rst=1'b0;
            //リセット解除 <== 時間単位100fsの場合に変更！
    #12000000 strt_msjt=1'b1;
            //測定有効 <== 時間単位100fsの場合に変更！
    #8100000 begin     // <== 時間単位100fsの場合に変更！
      $fdisplay(mcd, "ジッタ量---平均値:%d",I3.rslt_av);
      $fdisplay(mcd, "ジッタ量---正側ピーク値:%d,
          負側ピーク値:%d",I3.mj_pk_max, I3.mj_pk_min);
      tmp_r1=I3.mj_rms2_i;
      tmp_r2=I3.hit_jtrms;
      $fdisplay(mcd, "ジッタ量---RMSの2乗値:%g",

(tmp_r1/tmp_r2));
      $fdisplay(mcd, "VCO生成クロック半周期---最小値:%d,
                       最大値:%d ,I3.tsh_min_gen_clk,
                                   I3.tsh_max_gen_clk);
      $fdisplay(mcd, "ルンゲクッタ計算で許容誤差を満足
                 しなかった回数:%d" ,I2.I2.hit_min_h);
      $fclose(mcd);
      $finish;
    end

  end

  always begin //入力データ生成用リファレンス・クロックの生成
    ref_clk=1'b0; #(ts_refclk/2);
    ref_clk=1'b1; #(ts_refclk/2);
  end

  always begin
       //VCOチェーン回路制御電圧値を書き込むためのクロック生成
    msjt_clk=1'b0; #5000;
                       // <== 時間単位100fsの場合に変更！
    msjt_clk=1'b1; #5000;
                       // <== 時間単位100fsの場合に変更！
  end

  always @ (msjt_clk) begin
       //VCOチェーン回路入力(制御)電圧値の書き込み
    $fdisplay(mcd, "%d  %g",$time, I2.I3.vin_rl);
       //1行目は時間,2行目はVCO入力
  end

endmodule
```